NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

AN IMPLEMENTATION IN PASCAL:
TRANSLATION OF PROLOG
INTO PASCAL

by

Ahmet Saraydin

June 1985

Thesis Advisor: Neil C. Rowe

Approved for public release; distribution is unlimited

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) An Implementation in Pascal: Translation of Prolog into Pascal		5. TYPE OF REPORT & PERIOD COVERED Master's Thesis; June 1985
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Ahmet Saraydin		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93943-5100		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, California 93943-5100		12. REPORT DATE June 1985
		13. NUMBER OF PAGES 160
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		15. SECURITY CLASS. (of this report)
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution is unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Logic programming, Translation, Backtracking and Relational Database.

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This thesis tries to find a mapping algorithm between Prolog and Pascal languages. For this purpose, a small subset of Prolog is chosen and translated into Pascal code. Also, the concept of logic programming and its practical application in the programming language Prolog, are discussed. The reader is expected to be familiar with Pascal and Prolog. Keywords and phrases: Logic programming, Translation, Backtracking and Relational Database.

Approved for public release; distribution is unlimited

An Implementation in Pascal: Translation of Prolog into Pascal

by

Ahmet Saraydin
Captain, Turkish Army
Turkish Army War School, 1970

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 1985

ABSTRACT

This thesis tries to find a mapping algorithm between Prolog and Pascal languages. For this purpose, a small subset of Prolog is chosen and translated into Pascal code. Also, the concept of logic programming and its practical application in the programming language Prolog, are discussed. The reader is expected to be familiar with Pascal and Prolog.

Keywords and phrases: Logic programming, Translation, Backtracking and Relational Database.

TABLE OF CONTENTS

LIST OF FIGURES

I. PROLOGUE

A. PROGRAMMING LANGUAGES

1. Conventional Programming Languages

> "Conventional programming languages are growing ever more enormous, but not stronger. Inherent defects at the most basic level cause them to be both fat and weak: their primitive word-at-a-time style of programming inherited from their common ancestor, the von Neumann computer, their close coupling of semantics to state transitions, their division of programming into a world of expressions and a world of statements, their irability to effectively use powerful combining forms for building new programs from existing ones, and their lack of useful mathematical properties for reasoning about programs." [Ref. 1]

2. Software Crisis and Ada

It is virtually a cliche to say there is a software crisis. This crisis in software production is far greater than the situation of the early 50's that led to the development of high level languages to relieve the burden of coding. The symptoms appear in the form of software that is nonresponsive to user needs, unreliable, excessively expensive, untimely, inflexible, difficult to maintain, and not reusable. There are many ways to improve things a little and they are being tried. But to achieve a fundamental jump in our programming capacity, we need to rethink what we are doing from the beginning.

A programming language shapes the way we think about the solutions to our problems. Ideally, we desire a language that leads us to systems that map directly to the problem

space and that helps us control the complexity of programming solutions. Is Ada such a language or is it born dead? It is time to listen to Hoare.

> "I have been giving the best of my advice to this project since 1975. At first I was extremely hopeful. The original objectives of the language included reliability, readability of programs, formality of language definition, and even simplicity. Gradually these objectives have been sacrificed in favor of power, supposedly achieved by a plethora of features and notational conventions, many of them, like exception handling, even dangerous. We relive the history of the design of the motor car. Gadgets and glitter prevail over fundamental concerns of safety and economy.
>
> And so, the best of my advice to the originators and designers of Ada has been ignored. I appeal to you, representatives of the programming profession in the United States, and citizens concerned with the welfare and safety of your own country and of mankind: Do not allow this language in its present state to be used in applications where reliability is critical, i.e., nuclear power stations, cruise missiles, early warning systems, anti-ballistic missile defense systems. The next rocket to go astray as result of a programming language error may not be an exploratory space rocket on a harmless trip to Venus: It may be a nuclear warhead exploding over one of our own cities." [Ref. 2]

B. TOWARDS A SOLUTION: FUNCTIONAL PROGRAMMING

Just as high level languages enabled the programmer to escape from the intricacies of a machine's order code, higher level programming systems can provide help in understanding and manipulating complex systems and components. We need to shift our attention away from the detailed specification of algorithms, towards the description of the properties of the packages and objects with which we build. A new generation of programming tools will be based on the attitude that what we say in a programming system should be primarily declarative,

not imperative. The fundamental use of a programming system is not in creating sequences of instructions for accomplishing tasks, but in expressing and manipulating descriptions of computational processes and the objects on which they are carried out.

An alternative functional style of programming is founded on the use of combining forms for creating programs. Functional programs deal with structured data, are often nonrepetitive and nonrecursive, are hierarchically constructed, do not name their arguments, and do not require the complex machinery of procedure declarations to become generally applicable. Combining forms can use high level programs to build still high level ones in a style not possible in conventional languages. [Ref. 1]

> "This style of programming, also known as applicative programming and value-oriented programming, is important for a number of reasons. First, functional programming dispenses with the ubiquitous assignment operation. As structured programming is often called 'goto-less programming.', so functional programming can be called 'assignmentless programming.'
>
> The second reason that functional programming is important is that it encourages one to think at higher levels of abstraction. This is because functional programming provides a mechanism (functionals) for modifying the behavior of existing programs and for combining existing programs.
>
> The third reason for the functional programming is that it provides a paradigm for programming large, parallel computers. As we begin to reach speed of light and other limitations on computer speed, we can expect to see computers that achieve higher speed by greater parallelism. Functional programming's absence of assignments, independence of evaluation order, and ability to operate on entire data structures provide paradigms for programming these machines.

The fourth reason is its applications in 'Artificial Intelligence' (AI). Currently most AI programming is done in LISP, a language which inspired much of the early work in functional programming. PROLOG is the newest AI programming language and has a central role in the Japanese Fifth Generation [FG] Computer Project, PROLOG is a functional programming language [See Figure 1.1 for a sample Prolog Program]. Further, since AI techniques are finding wider and wider applications, functional programming is important to more than just AI programmers: it is important to all programmers.

The fifth reason that functional programming is important is that it is valuable for developing executable specifications and prototype implementations. The simple underlying semantics and rigorous mathematical foundations of functional programming along with its high expressive ability make functional programming an ideal vehicle for specifying the intended behavior of programs. Functional programming can serve this function even if no functional programming language system is available to execute the program. However, if such a system is available then we have something very valuable: an executable specification. This can be used as a prototype implementation to determine if the specifications are correct, and as a benchmark against which later implementations can be compared. Thus, even if the reader never intends to write do functional programming, it can still be a valuable tool for the formulation, expression and evaluation of program specifications.

Finally, functional programming is important because of its connections to computer science theory. Functional programming provides a simpler framework for viewing many of the decidability questions of programming and computers than do the usual approaches." [Ref. 3]

C. THE FIFTH GENERATION COMPUTER PROJECT

In April 1982, Japan launched a research project to develop computer systems for the 1990's. The project, called the Fifth Generation computers project, will span 10 years. Its ultimate goal is to develop integrated systems, both hardware and software, suitable for the major computer application in the next decade, identified by the Japanese as

```
produce(X,Y,Z,U) :- prod1(X,Y,Z,U,[ ]).

prod1(Clist,STM,Rslt,[Rname|Plan],Hist)
            :-recognize(Clist,STM,Rname,Action),
              control_test([Rname|Hist]),
              act(Action,STM,NewSTM),
              prod1(Clist,NewSTM,Rslt,Plan,[Rname|Hist]).
prod1(Clist,Rslt,Rslt,[ ],Hist).

recognize(Clist,STM,Rname,Action)
            :-prod_rule(Class,Rname : Cond => Action),
              member(Class,Clist),
              hold(Cond,STM).

hold([ ],STM).
hold([C|CL],STM) :- holdeach(C,STM),!,hold(CL,STM).

holdeach(absent(X),[ ]).
holdeach(absent(X),[Fact|STM])
            :- not(X = Fact),holdeach(absent(X),STM).
holdeach(X = Y,STM) :- x = Y.
holdeach(found(X),STM) :- holdeach(X,STM).
holdeach(X,[ ]) :- call(X).
holdeach(X,[X|STM]).
holdeach(X,[X|STM]) :- holdeach(X,STM).

act([ ],STM,STM).
act([Act|AL],STM,New_STM)
            :- acteach(Act,STM,Int_STM),!,
               act(AL,Int_STM,New_STM).

acteach(delete(X),[ ],[ ]).
acteach(delete(X),[X|Y],Y).
acteach(delete(X),[Y|L],[Y|L1])
              :- acteach(delete(X),L,L1).
acteach(insert(X),L,[X|L]).
acteach(replace(X,Y),[ ],[ ]).
acteach(replace(X,Y),[X|L],[Y|L]).
acteach(replace(X,Y),[Z|L],[Z|L1])
            :- acteach(replace(X,Y),L,L1).
acteach(Else,STM,STM) :- call(Else).
```

Figure 1.1 A Simple PRODUCTION SYSTEM Written in PROLOG

"knowledge information processing." Even though it may ultimately have applicable results, the current focus of the project is basic research rather than the development of commercial products. [Ref. 4]

In addition to bringing Japan into a leading position in the computer industry, the project is expected to elevate Japan's prestige in the world. It will refute accusations that Japan is only exploiting knowledge imported from abroad without contributing any of its own to benefit the rest of the world. Hence, the project aims at original research and plans to make its results available to the international research community. [Ref. 5]

The most intriguing aspect of the project is its commitment to build the Fifth Generation systems around the concepts of logic programming. In the following paragraphs we trace the roots and rationale for this commitment.

There are many attributes that prescribe a computer system; however, the most important one is what language we accept as the main programming language. For application areas, the basic structure of software systems and the frame of computer architecture are all determined by this language. So in this project, this main programming language, FG-Kernel Language, seems to be the most important research theme. The research and development of this language must be carefully pursued on the basis of systematic studies on various aspects such as artificial intelligence (problem solving and knowledge

representation), software engineering, examination of various programming language proposed, etc. The reasons why a logic programming language (PROLOG) is chosen as the kernel of FG-Kernel language are summarized below.

It is appropriate for programming of knowledge information processing system. List processing, database mechanism similar to relational database, pattern matching (unification) which clearly represents the composition and decomposition of data structure and database research, non-deterministic processing, etc. are indispensable processing functions in programming of knowledge information processing systems. PROLOG has all basic parts of these functions, and moreover, is able to be extended to get more high performance functions.

It gives new paradigms of programming. A non-procedural representation scheme, high modularity, a happy blending of computation and database search etc. are new programming paradigms. These paradigms, what is better still, make it much easier to deal with programs and programming as formal objects and give great possibilities to realize a program verifier and an automatic programming system.

It succeeds to the results of efforts made by current programming languages. Much has been discussed about the relationship between logic programming languages and functional languages, and it has become generally appreciated that these languages will play the leading part in future programming. To be concrete, also as to Lisp, the functional language that is most widely put into practical use at

present, it is possible to extend PROLOG efficiently to include useful functions of Lisp as a subset. PROLOG can put a search mechanism with backtracking control into practical use by using logical formulas (Horn clauses) as language constructs and by improving implementation techniques.

It introduces new computer architectures. FG-Kernel Language will be first implemented on an conventional large scale computer and then on a high performance personal computer. According to the research plan of the Fifth Generation Computers, the language will be improved and extended step by step, based on actual experience and various research results. And finally, the language will become a machine language for the target machine of this project. Consequently, the language (Edinburgh version) must be such a language as fundamentally has all of the appropriate mechanisms for data flow machines and data base machine architectures supposed as basic architectures of the target machine. PROLOG has a great possibility for this, too.

For the above reasons, PROLOG has been chosen as the kernel of Kernel Language. Next, the main features of improvement and extension of PROLOG now under study are enumerated. We give priority to the arrangment of all primitive and necessary functions over invention of high level ones.

1. <u>Abstract Data Types (Encapsulation)</u>

The usefulness of abstract data types has been well known and recently most new programming languages have adopted

it as the basic function. But the current version of PROLOG doesn't have this construct explicitly. So, we have to introduce it in natural way. To introduce every function of the abstract data type and to make clear its function for program specification and program verification remain as a long term research theme.

It is desired that this extension is made by natural enlargement of functions which PROLOG has now. PROLOG has one internal database. In this database, all clauses (unit and non-unit) are stored. There are predicates which assert and retract these clauses, and the way to cause side-effects is to alter the contents of the database with these predicates. This situation can be interpreted as follows: there is only one abstract data type called internal database. Consequently, to make it possible to define a number of abstract data types is to make it possible to create a number of databases, which can be called Micro databases. Various advantages are obtained by the introduction of Micro databases. For instance,

(a) Side-effects are localized.

(b) Structures are introduced into programs. If a nested structure is permitted among databases, more complicated program structures can be represented.

(c) Separate compilation becomes available. Clauses which are not exported, are never accessed from the outside. So, it is possible to compile calling sequences (unification) to these clauses.

2. Refined Higher-order Extensions

PROLOG is a simple and powerful language based on first-order logic. For practical use, however, various higher-order extensions have to be introduced. What is essential is still open to discussion. For example, it is said that higher-order extensions like lambda expressions and predicate variables are not very essential and first-order logic has enough ability. In Lisp, for example, the most primitive mechanism for higher-order programming is that program and data have the same structure, and that quote and eval functions are provided, which control whether some data structures are regarded as program or data. This mechanism is introduced to PROLOG too as a primitive one. The basic data structure of Lisp is the list (s-expression). To PROLOG, the tuple is regarded as a basic one. Each term, predicate and Horn clause is able to be internally represented as a tuple. At the head of each tuple, the tuple name is placed and the attribute of this name indicates what the tuple represents. And then, for composition and decomposition of tuples, unification is extended and some predicates are introduced.

The most fundamental construct for the control structure of PROLOG is the cut operation. This operation is very powerful, but its effect is very hard to understand. So, it is compared to the goto statement in a conventional language. It is possible to introduce more structured constructs for control and banish the cut operation, as we did the goto

statement. For example, the introduction of a selection mode for clauses is possible.

3. Enough Preparation of Programming Tools

Evaluation with backtracking makes debugging very difficult. This means it is necessary to prepare more powerful tools. These include: (1) Debugger, which traps evaluation by error or break, keeps the environment as it is and responds to various users' commands, (2) Tracer, which traces the history of evaluation of specified predicates and variables and displays it in pretty format, (3) Stepper, which evaluates program steps one by one and displays various states by the minute, (4) Editor, which edits clauses with pattern matching, etc. These tools are combined into one total programming system in order to be invoked at any place.

4. High Level Data Structure

It is pointed out that data structures such as sets and bags which collect elements to satisfy certain conditions, represented by predicates, are improtant. For this, the most primitive higher-order predicate is provided to PROLOG as well.

5. Useful Functions for System Description

Interpreters, compilers, file systems, tools for debugging, etc., a lot of system programs have to be developed. The Kernel part of them can be implemented by micro programs. The rest are desirable to be implemented by PROLOG itself. For this purpose, it is possible to introduce efficient system description functions into it. For example, they are:

Abstract data types with good efficiency. A compiler is able to transform the Micro database introduced in (1) into very efficient object codes under a certain restriction. For example, it transforms a clause in Micro database into such codes as fetch and store terms directly in a predicate which represents its internal states.

Refined system data structures. Data structures which represent the internal state of the system are refined. Basic predicates which access and manipulate them and basic protection mechanism are both provided.

Constructs for parallel processing. Necessary parallel processing constructs for programs controlling external devices are introduced as simply as possible.

Compared with an ordinary system description language, PROLOG has far higher level functions, therefore, it is apt to be thought that it is not appropriate for system description. But, under natural restrictions and degeneration of functions, it is able to guarantee the same efficiency as an ordinary system description language does. Examples of these restrictions are: There is no non-deterministic selection. Unification is restricted. A term is a variable or a constant. Furthermore, it is restricted to the parameter binding of an ordinary functional language.

6. The Others

Besides the above, the following functions have to be researched. They are: Large scale databases, connection with external databases (relational databases), other search modes different from top-down and depth-first search, and the improvement of backtracking search mechanism.

II. EXPRESSION OF RELATIONAL DATABASE QUERIES IN LOGIC

A. RELATIONAL DATABASES

Development of data base systems was one of the core elements during the progress in the 70's of computer technology. How to organize and how to utilize gigantic volumes of data were the questions. The progress was made by accumulating experience. Along with it, efforts to organize such experience theoretically also went on.

Codd's proposal for relational databases was made early in the 70's, but is only now about to become a major stream in structuring data bases. This is based on a theory of "relations". As query languages for the data bases predicate formulas (relational calculus) and functional formulas (relational algebra) are proposed. These are mutually interchangeable. They can be regarded as certain kinds of special logics, and through the 70's a great deal of theoretical research effort was made in this area.

B. QUERIES AND LOGIC

Relational database retrieval is viewed as a special case of deduction in logic. It is argued that expressing a query in logic clarifies the problems involved in processing it efficiently (query optimization). We want to describe a simple way for defining a query so that it can be executed by the elementary deductive mechanism provided in the programming language PROLOG.

Several current relational database formalisms have a core which can be viewed as no more than a syntactic variant of a certain subset of logic. To illustrate this, let us consider an example written in Quel.

```
range of E,M is employee
range of D is dept
retrieve (E.name)
where E.salary > M.salary
and E.manager = M.name
and E.dept = D.dept
and D.floor = 1
and E.age > 40
```

In ordinary English, this query means: "Which employees aged over 40 on the first floor earn more than their managers?" This query refers to relations:

```
employee(name,dept,salary,manager,age)
dept(dept,floor)
```

This query can be expressed in logic (using Prolog oriented syntax) as:

```
answer(E) :- employee(E,D,S,M,A),
             A > 40,
             dept(D,1),
             employee(M,_,S1,_,_),
             S > S1.
```

Read this as:

E is an answer if

E is an employee, dept D, salary S, manager M, age A, and

A is greater than 40 and

D is a department on floor 1 and

M is an employee, salary S1, and

S is greater than S1.

Here the identifiers starting with a capital letter, such as E, D, S, etc., are logic variables, which can be thought of as standing for arbitrary objects of the domain. Contrast this with the variables of Quel, which denote arbitrary tuples of a certain relation specified in a range statement. (Because, in this example, tuples can be uniquely identified by their first fields, it is natural for the logic variable corresponding to this field to have the same name identifier as is used for the tuple variable in the Quel version). For each tuple variable in a Quel query, there is, in the logic version, a corresponding goal (also called "atomic formula"), e.g.,

dept(D,1)

A goal consists of a predicate, naming the range relation of the corresponding tuple variable, applied to some arguments, corresponding to the fields of this relation. Quel constraints which are identities map into an appropriate choice of variables or constants (such as '1') for certain goal

arguments. This aspect tends to make the logic form of the query more concise and, it can be argued, easier to comprehend. Note the use of '_' to denote an "anonymous" variable, which is only referred to once, and which therefore does not need to be given a distinct name. Quel constraints which are inequalities map into separate logic goals. The Quel query as a whole maps into a restricted kind of implication, called a clause, where the target of the query appears as the conclusion of the implication (to the left of the ':-').

Clauses can be used not only to represent queries, but also to express the information which makes up the database itself. (It is this aspect which distinguishes what will be described here from much other work relating logic and databases).

In general a clause consists of an implication, which in the Prolog subset of logic is restricted to the form:

```
P :- Q1, Q2, ... Qn.
```

meaning "P is true if Q1 and Q2 and ... Qn are true", where P and the Qi may be any goals. If n = o, we have what is called a unit clause, which is written simply as:

```
P.
```

meaning "P is true".

For example, here are some unit clauses, representing elementary facts, which serve to define which tuples make up relation 'parent'.

```
parent(david,hugh).
parent(david,winifred).
parent(ben,david).
parent(ben,jane).
```

The first clause, for instance, may be read as:

"David has a parent Hugh".

Here we have defined a database relation by explicitly enumerating its tuples. However it is also possible to define a relation implicitly, through general rules expressed as non-unit clauses. For example, here is the definition of the 'ancestor' relation in terms of the 'parent' relation:

```
ancestor(X,Z) :- parent(X,Z).
ancestor(X,Z) :- parent(X,Y),ancestor(Y,Z).
```

Read these clauses as:

"X has an ancestor Z if X has a parent Z".

"X has an ancestor Z if
X has a parent Y and Y has an ancestor Z".

Note that the second clause makes the definition recursive. We can think of 'ancestor' as a "virtual" relation. A pair <X,Y> belongs to the 'ancestor' relation if:

```
ancestor(X,Y)
```

is a logical consequence of the clauses which make up the database. Thus one can infer, for example, that one of Ben's ancestors is Hugh, i.e.,

```
ancestor(ben,hugh)
```

This use of logic clauses to define a database gives much greater power and conciseness than is available in most conventional relational database systems. These systems do not allow an equivalent recursive definition of the 'ancestor' relation, for example.

In fact, the logic subset we have been looking at forms the basis of a general purpose programming language, Prolog. A Prolog system is essentially a machine which can generate solutions to a problem by enumerating all instances of some goal which are valid inferences from the clauses which make up a "program". For example, if the user presents the query:

```
answer(X) :- ancestor(ben,X).
```

Prolog responds with the following list of possible values for X, representing all the ancestors of Ben that can be deduced:

```
X = david; X = jane; X = hugh; X = winifred
```

The solutions are in fact produced in exactly this order. How this takes place will not be described.

In Prolog, the ordering of clauses in a program, and the ordering of goals in the right-hand side of a clause, provide important control information, which helps to determine the way a program is executed.

To execute a goal (such as 'ancestor(ben,X)' in the previous query), Prolog tries to match it against the left-hand side of some clause, by finding values for variables which make the clause "head" identical with the goal. When

successful, Prolog then recursively executes the goals (if any) in the right-hand side of the clause, which will by now have been modified by the results of the matching. When no match can be found, or when there are no more goals left to execute, Prolog backtracks. That is it goes back to the goal most recently matched, undoes the effects of the match, and then seeks an alternative match.

Clauses are tried for a match in the order they appear in the program. Goals in the right-hand side of a clause are executed in the order they appear in that clause. The matching process is actually unification, a process which effectively produces the least possible instantiation of variables necessary to make the two goals identical.

Prolog's backtracking can be thought of as a generalized form of iteration. Thus the two clauses for 'ancestor', when used to satisfy a goal such as 'ancestor(ben,X)', give a behaviour when executed by the Prolog equivalent to the following procedure:

```
To generate Zs who are ancestors of X:
   first generate Zs who are parents of X;
   then for each Y who is a parent of X:
      generate Zs who are ancestors of Y.
```

In fact, some compilers can compile such clauses into code which is comparable in efficiency with iterative loops in a more conventional language.

As a final remark, one should note that the Prolog subset of logic includes, besides the variables and elementary constants seen so far, objects which are structures. In this respect, while being similar to many other programming languages, it is a further important generalization of most relational database formalisms.

In fact, Prolog was not designed with relational database retrieval in mind, it was conceived purely as a programming language. The efficiency of processing of Prolog queries may be discussed. The Prolog-based approach of Chat-80 compares with the strategies used in conventional relational database systems. [Ref. 6]

III. TRANSLATION OF A SUBSET OF PROLOG INTO PASCAL

A. PASCAL AS AN IMPLEMENTOR LANGUAGE

Pascal is chosen as an object language for this application, because it does have some excellent features. [Ref. 7] Here is a list of positive aspects:

1) small number of well-chosen keywords,
2) small number of syntax and semantics rules,
3) meaning of Pascal instructions is highly independent of environment, which promotes portability of programs,
4) excellent data structuring methods,
5) clean and efficient control structuring,
6) excellent for programming "in the small",
7) gives a feeling of reliability,
8) with some care, readability can be kept high.

Pascal is definitely very useful in the following areas:

1) compiler writing, cross assemblers and compilers,
2) text processing,
3) general, off-line utility programs (editors, etc.),
4) treatment of non-numerical data,
5) processing of trees, lists and other complex data structures,
6) some mathematical problems,
7) construction of portable programs.

We do not want to deal with the existing problems in that language. This is beyond the scope of this thesis.

B. PROLOG AND BACKTRACKING

Prolog is a simple but powerful programming language founded on symbolic logic. The basic computational mechanism is a pattern matching process ("unification") operating on general record structures ("terms" of logic). It can be argued that pattern matching is a better method for expressing operations on structured data than conventional selectors and constructors--both for the user and for the implementor. From a user's view the major attraction of the language is ease of programming. Clear, readable, concise programs can be written quickly with a few errors.

Prolog has many parallels with Lisp. Both are interactive languages designed primarily for symbolic data processing. Both are founded on formal mathematical systems--Lisp on Church's lambda calculus, prolog on a subset of classical logic. Like pure Lisp, the Prolog language does not (explicitly) incorporate the machine-oriented concepts of assignment and references (pointers). Furthermore, pure Lisp can be viewed as a specialization of Prolog, where procedures are restricted to simple functions and data structures are restricted to lists.

Prolog differs from most programming languages in that there are two quite distinct ways to understand its semantics. The procedural semantics is the more conventional, and describes in the usual way the sequence of states passed through when executing a program. In addition a Prolog

program can be understood as a set of descriptive statements about a problem.

The declarative semantics which Prolog inherits from logic provides a formal basis for such a reading. It simply defines (recursively) the set of terms that are asserted to be true according to a program. A term is true if it is head of some clause instance and each of the goals (if any) of that clause instance is true, where an instance of a clause (or term) is obtained by substituting, for each of zero or more variables, a new term for all occurrences of the variable.

The procedural semantics describes the way a goal is executed. The object of the execution is to produce true instances of the goal. It is important to notice that the ordering of clauses in a program, and goals in a clause, which are irrelevant as far as the declarative semantics is concerned, constitute crucial control information for the procedural semantics.

To execute a goal, the system searches for the first clause whose head matches or unifies with the goal. The unification process finds the most general common instance of two terms, which is unique if it exists. If a match is found, the matching clause instance is then activated by executing in turn, from left to right, each of the goals of its body (if any). If at any time the system fails to find a match for a goal, it backtracks, i.e., it rejects the most recently

activated clause, undoing any substitutions made by the match with the head of the clause. Next it reconsiders the original goal which activated the rejected clause, and tries to find a subsequent clause which also matches the goal.

Prolog owes it simplicity firstly to a generalization of certain aspects of other programming languages, and secondly to omission of many other features which are no longer strictly essential. This generalization gives Prolog a number of novel properties. We shall briefly summarize them.

1) General records structures take the place of Lisp's S-expressions. An unlimited number of different record types may be used. Records with any number of fields are possible, giving the equivalent of fixed bound arrays. There are no type restrictions on the fields of a record.

2) Pattern matching replaces the use of selector and constructor functions for operating on structured data.

3) Procedures may have multiple outputs as well as multiple inputs.

4) The input and output arguments of a procedure do not have to be distinguished in advance, but may vary from one call to another. Procedures can be multi-purpose.

5) Procedures may generate, through backtracking, a sequence of alternative results. This amounts to a high level of iteration.

6) Unification includes certain features which are not found in the simpler pattern matching provided by some languages. One can sum this up in the equation: Unification = pattern matching + the logical variable.

8) The characteristics of the "logical" variable are as follows. An "incomplete" data structure (i.e., containing free variables) may be returned as a procedure's output. The free variables can later be filled in by other procedures, giving the effect of

implicit assignments to a data structure. Where necessary, free variables are automatically linked together by "invisible" references. As a result, values may have to be "dereferenced". This is also performed by the system. Thus the programmer need not be concerned with the exact status of a variable--assigned or unassigned, bound to a reference or not. In particular, the occurrences of a variable in a pattern do not need any prefixes to indicate the status of the variable at that point in the pattern matching process. In short, the logical variable incorporates much of the power of assignment and references in other languages. This is reminiscent of the way most uses of goto can be obviated in a language with well structured control primitives.

9) Program and data are identical in form. Clauses can usefully be employed for expressing data.

10) There is a natural declarative semantics in addition to the usual procedural semantics.

11) The procedural semantics of syntactically correct program is totally defined. It is impossible for an error condition to arise or for an undefined operation to be performed. This is a contrast to most programming languages. A totally defined semantics ensures that programming errors do not result in bizarre program behaviour or incomprehensible error messages.

C. A SUBSET OF PROLOG (SPROLOG)

For the purpose of this work: we select a small subset of Prolog and we will call it Small Prolog (SPROLOG). This subset only includes some primitive data structures, such as atoms and integer numbers. The formal definition of this language is given in Figure 3.1.

SPROLOG also has some restrictions. These are:

1) There is no anonymous (_) variable. This restriction eliminates the possibility of violation of procedure naming rule in Pascal,

2) Recursive definition is not allowed,

3) Only nonnegative integer numbers can be handled,

```
<sprolog>          ::= <rule or fact>  {<rule or fact>}
<rule or fact> ::= <rule> | <fact>
<rule>             ::= <head> :-  <body> .
<fact>             ::= <head> .
<head>             ::= <prefix>
<body>             ::= <structure>  { , <structure>}
<structure>        ::= <infix> | <prefix>
<infix>            ::= <asg> | <expression>
<asg>              ::= <variable> is <arithmetic>
<expression>       ::= <relational> | <arithmetic>
<relational>       ::= <arithmetic> <rel operator>
                       <arithmetic>
<arithmetic>       ::= <variable or number>
<arithmetic>       ::= <variable or number> <art operator>
                       <variable or number>
<prefix>           ::= <procname>
<prefix>           ::= <procname> ( <variable or constant>
                         { , <variable or constant>} )
<procname>         ::= <small>
<procname>         ::= <small> <letter or digit>
                        {<letter or digit>}
<variable or number>   ::= <variable> | <number>
<variable or constant> ::= <variable> | <constant>
<variable>         ::= <capital>
<variable>         ::= <capital> <letter or digit>
                        {<letter or digit>}
<constant>         ::= <atom> | <number>
<atom>             ::= <small>
<atom>             ::= <small> | <letter or digit>
                        {<letter or digit>}
<number>           ::= <digit>   {<digit>}
<letter or digit> ::= <letter> | <digit>
<letter>           ::= <capital> | <small>
<capital>          ::= A|B|C|D|E|F|G|H|I|J|K|L|M
                       N|O|P|Q|R|S|T|U|V|W|X|Y|Z
<small>            ::= a|b|c|d|e|f|g|h|i|j|k|l|m
                       n|o|p|q|r|s|t|u|v|w|x|y|z
<digit>            ::= 1|2|3|4|5|6|7|8|9|0
<rel operator> ::= <  |  >  |  =  |  <>
<art operator> ::= +  |  -  |  *  |  /
```

Figure 3.1   SPROLOG in BNF Form

4) Any variable or atom may have at most ten characters,

5) Any program must have only one query clause which is defined as the last rule of the program,

6) Any predicate name placed in the body clause must have been declared before as a head clause of a rule. This eliminates taking into consideration the "forward" declarations inherited in Pascal,

7) Arithmetic expressions may have at most one operator.

These restrictions make this implementation easy. But, we lose the beauty of the problem.

D. DESIGN

We will develop our work by using the following example. Suppose we have the Prolog program illustrated in Figure 3.2. Our job is to translate it to a Pascal program. We consider that all head clauses of Prolog correspond to the function declarations in Pascal. That is, "pop", "area", "density", "ans" and "query" are all names of the functions which will be called by the calls that are placed in the body clauses anywhere inside the program. The type of these functions is always boolean. If the body clause does not exist, this means that this function will not call any other functions.

```
pop(china,825).
pop(india,586).
area(china,3380).
area(india,1139).
density(C,D):-pop(C,P),area(C,A),D is P/A.
ans(C1,D1,C2,D2):-density(C1,D1),density(C2,D2),
                  D1>D2,20*D1<21*D2.
query:-ans(X,Y,Z,T).
```

Figure 3.2 Sample PROLOG Program

The transfer of parameters defined in the Prolog program will cause a little problem, because Prolog does not force the programmer to declare them with the same number and the same type. For example, "density" might be declared with many number of parameters in various places in the program. This leads us to use pointer variables that point to the formal and actual parameters which are stored in the storage area. This idea facilitates parameter passing among functions without using variant record declarations and also prevents the probable translation errors which may result from some features of Pascal, such as "strong typing" or "type checking".

We need also to inform the callee about the caller's name for the following reasons. As shown in the sample program in Figure 3.2, the same name may refer to several callees which may have different numbers and types of parameters. This information will provide a basis for the matching and binding processes. So, to implement this idea, we will enumerate the names of functions and their parameters in the following simple way.

In Prolog source code, enumerate all names from top to bottom and from left to right. In the same way, give also a sequence number to all parameters. So, in the above example, "pop" will have number 1 and the last name "ans" will be numbered as 15. Also, the actual parameter of the first "pop" clause, which is "china", will be the first parameter of this

program and the formal parameter "T" of the "ans" will get number 38. Notice that "is", ">" and "<" in the program are not user defined functions. These are predefined and we will use them from the library.

We already have some problems. There exists more than one alternative clause for the names "pop" and "area". It is impossible to declare two functions with the same name in Pascal. To solve it, we rename the first "pop" as "popl" and the second one as "pop2". Also, we need to define another function whose name is "pop" which will drive all the alternatives according to a logical sequence. This process will be applied to all functions which have alternative clauses. We continue our example in the following tables.

The first table ("Procedure Table") includes some information about the functions (see Figure 3.3). The leftmost column is the function number. This number will be used during the execution phase, when needed, to identify any function. The second column shows the name of tne functions.

seq. No.	function Name	parameter pointers		alternative pointers	
1	pop	1	2	0	0
2	pop	3	4	0	0
3	area	5	6	0	0
4	area	7	8	0	0
5	density	9	10	0	0
6	pop	11	12	1	2
7	area	13	14	3	4
8	is	15	18	0	0
9	ans	19	22	0	0
10	density	23	24	0	0
11	density	25	26	0	0
12	>	27	28	0	0
13	<	29	34	0	0
14	query	0	0	0	0
15	ans.	35	38	0	0

Figure 3.3 Procedure Table

But, ">" or "<" can not be legal Pascal function names. Later, we can change them to "greater", "lessthan", etc. The third and fourth columns are all pointers. They point to the "Parameter Table" (see Figure 3.4) for the associated parameters of that function. Because the function "query" does not have any parameters, its parameter pointers do not point to anything. On the other hand, the last column shows the alternative clauses of that function. For example, the functions "pop" and "area" have two non-zero alternative pointers. In other words, this means that these functions have two alternatives.

The information about parameters is shown in the Parameter Table (see Figure 3.4). The parameter type represents the type of the parameter. Variables, integers and atoms will have the numbers 1, 2 and 3, respectively. However, other numbers which are greater than 3, indicate the existence of arithmetic expressions. The fourth column of the table points to the associated function for those parameters.

The last table (see Figure 3.5) renames the alternative clauses. If we have several functions with the same name, we rename then and then we will be able to use them with these names. In fact, these three tables are not so simple as shown in the figures. The reader may refer to the sample programs given in the appendices.

seq. No.	parameter Name	parameter type	pointer to proc table
1	china	3	1
2	825	2	1
3	india	3	2
4	586	2	2
5	china	3	3
6	3380	2	3
7	india	3	4
8	1139	2	4
9	C	1	5
10	D	1	5
11	C	1	6
12	P	1	6
13	C	1	7
14	A	1	7
15	D	1	8
16	P	1	8
17	/	10	8
18	A	1	8
19	C1	1	9
20	D1	1	9
21	C2	1	9
22	D2	1	9
23	C1	1	10
24	D1	1	10
25	C2	1	11
26	D2	1	11
27	D1	1	12
28	D2	1	12
29	20	2	13
30	*	9	13
31	D1	1	13
32	21	2	13
33	*	9	13
34	D2	1	13
35	X	1	15
36	Y	1	15
37	Z	1	15
38	T	1	15

Figure 3.4 Parameter Table

seq. No.	function Name	pointers to proc table	
1	pop1	1	2
2	pop2	3	4
3	area3	5	6
4	area4	7	8

Figure 3.5 Alternative Clauses Table

E. MEMORY MANAGEMENT AND PROBLEMS

All variables and constants may be handled by using the dynamic storage feature of Pascal. It seems necessary to describe four kinds of records to keep a parameter in a heap area.

The first record ("Procedure Record") contains enough information about the rule number, function number, and parameter number. Also, its last item points to the "Parameter Specification Record". This record keeps the parameter type, parameter name, if any, and it also has a cell pointer which indicates the related Cell. A Cell is itself a pointer which points to the "Value Record". This record saves the value of that parameter. The last one has to have the variant record specification to store various types of parameters. If a parameter does not have value, namely an uninitialized variable, the Cell pointer will not show any "Value Record".

To handle arithmetic expressions, the Cell pointer will point to the associated binary tree for that expression. The

leaves of the tree are also pointers that point to the related "Parameter Specification Record". Also, the same idea can be applied to the list data structures, because it is possible to represent the list as a binary tree.

The variables that are local to a rule will share the same storage area via the "Specification Pointer" defined in its "Procedure Record". This is also true for all the constants of the Program. The same constants, like "china", will be stored only once. The associated cell pointers will provide the way for the common storage.

To bind a value to a variable, the Cell pointer of this variable will point to a "Value Record" which is determined at the time of matching process. This process will create a long chain during the execution of the program. Also, the reverse process is necessary when backtracking and resatisfying occurs. At this point, our design and, finally, this thesis is completely unsuccessful. Due to the storage management and the complexity of execution phase, we restrict again SPROLOG so that our implementation will only be able to execute the "facts" and one rule which is defined at the end of the Prolog program. In this case, this implementation will be useful to define and implement relational databases and query applications. (Our implementation allows processing at most 99 different relations).

Now we are ready to translate the sample Prolog program given in Figure 3.2 into Pascal. Function "pop" and its two

alternatives are shown in Figure 3.6. Formal parameters "a" and "i" which are defined as integers, are function numbers. The parameter "a" is the number of the caller as described in Figure 3.3. The other parameter "i" corresponds to the callee's number which is driven in the "pop" function by the "case" statement. The function of the "case" statement placed in "pop" is very important. All alternatives clauses will be tried by this construction until the "resatisfaction" is not required any more or any impossible condition occurs.

The "match" function included in "popl" and "pop2" is the library function. The unification and binding process will be made by this function. If its returned value is true, this means that the "binding" occurred after the "matching" process.

The function "area" and its alternatives "areal" and "area2" are shown in Figure 3.7. These functions have been constructed with the same way as in the example "pop".

Before describing the other functions, we want to note the importance of "accept" function shown in Figure 3.8. This is the general driver for all functions. It accepts any function name and its number as arguments and calls all possible alternative functions. For example, to call "popl" or "pop2", "accept" creates functions numbers which will be used by the "case" statement of the "pop". If any returned value is "true" during the execution of "for" loop, "accept" will also return a "true" value. As you noticed, the first

```
function pop1(a,i:integer):boolean;
begin
    pop1:=match(a,i);
end;
function pop2(a,i:integer):boolean;
begin
    pop2:=match(a,i);
end;
function pop(a,i:integer):boolean;
begin
    case i of
    1:pcp:=pop1(a,i);
    2:pcp:=pop2(a,i);
    end;
end;
```

Figure 3.6 Function POP

```
function area3(a,i:integer):boolean;
begin
    area3:=match(a,i);
end;
function area4(a,i:integer):boolean;
begin
    area4:=match(a,i);
end;
function area(a,i:integer):boolean;
begin
    case i of
    3:area:=area3(a,i);
    4:area:=area4(a,i);
    end;
end;
```

Figure 3.7 Function AREA

```
function accept(function name(a,j:integer)
        :boolean;a,j:integer):boolean;
var i:integer;
begin
    for i:=first(a) to last(a) do
    begin
        if(first(a)>last(a)) then leave;
        if(name(a,i)) then
        begin
            proc(.a.).now:=succ(i);
            accept:=true;
            return;
        end;
    end;
    accept:=false;
    resetit(a);
end; (* accept *)
```

Figure 3.8 Function ACCEPT

job of "accept" is to try all alternative clauses. If there are no more alternatives to be resatisfied, it returns "false". The functions "first" and "last" determine the function numbers of alternatives for any caller function. The function "resetit" will reset the numbers of alternatives for the future use.

The structure of the function "density" (see Figure 3.9) summarizes the resatisfying and backtracking processes inherited in the Prolog program. If there is any "resatisfaction" request, the execution sequence has to start from the rightmost clause to leftmost clause of the Prolog program. Also, if there is a need for the backtracking, this process also will begin from the right to the left.

The logical variable "resatisfy" in the "density" function is a global variable to the program. Its job is to

```
function density(a,i:integer):boolean;
label 6,7,8,9,99;
begin
    if resatisfy then
    begin
       break(8);
       goto 8;
    end;
    6:if (accept(pop,6,a)) then goto 7;
       goto 99;
    7:if (accept(area,7,a)) then goto 8;
       if not(possible(6)) then goto 99;
       break(6);
       goto  6;
    8:if (accept(is,8,a)) then goto 9;
       if not(possible(7)) then goto 99;
       break(7);
       goto  7;
    9:if okay(a) then
       begin
          density:=true;
          return;
       end;
    99:density:=false
end;
```

Figure 3.9 Function Density

determine if the context of "resatisfaction" exists. If it does, then the existing links for binding variables are broken by the "break" and transfer goes to the last function corresponding to the last clause of the Prolog program. In the "density" example, transfer will go to statement labeled 8, if the "resatisfaction" occurs. This transfer will cause the function "is" to be called.

All "goto's" in the "density" function simulates the "backtracking" process of Prolog. As noticed, after trying all possibilities for the "pop" function, transfer goes to the last statement of the "density" function. Otherwise, if any alternative of "pop" returns the "true" value, then

transfer passes to satisfy the next function corresponding to the next clause in the Prolog program. If this function can not create a "true" value, now the "backtracking" process begins. The transfer goes to the last tried function, if the last one has already any alternative to be satisfied. This checking is made possible by the "possible" function.

The execution sequence may reach to the last "if" statement (in "the density example, the statement labeled 9). The function "okay" checks the returned values of called functions in that function (namely, in the "density" example. They are "pop", "area" and "is"). Finally, it evaluates them and causes to be assigned a truth value to that function.

The function "ans" (see Figure 3.10) is also created by the same logic described before. It calls some system functions such as "greater" and "lessthan". These correspond to the Prolog clauses which contains the relational operators, ">" and "<", accordingly.

The function "query" (see Figure 3.11) corresponds to the Prolog query given by the user. Its construction is not different from the other functions described so far. The actual execution chain starts from this point. Eventually, the value of this function will be the answer to the user.

Finally, the main body of the Pascal program is illustrated in Figure 3.12. Its important feature is to demonstrate the starting point of the resatisfying process. The user may request to resatisfy his goal, namely he enters ";".

```
function ans(a,i:integer):boolean;
label 10,11,12,13,14,99;
begin
    if resatisfy then
    begin
        break(13);
        goto 13;
    end;
    10:if (accept(density,10,a)) then goto 11;
       goto 99;
    11:if (accept(density,11,a)) then goto 12;
       if not(possible(10)) then goto 99;
       break(10);
       goto  10;
    12:if (accept(greater,12,a)) then goto 13;
       if not(possible(11)) then goto 99;
       break(11);
       goto  11;
    13:if (accept(lessthan,13,a)) then goto 14;
       if not(possible(12)) then goto 99;
       break(12);
       goto  12;
    14:if okay(a) then
       begin
           ans:=true;
           return;
       end;
    99:ans:=false
end;
```

Figure 3.10 Function ANS

```
function query(a,i:integer):boolean;
label 15,16,99;
begin
    if resatisfy then begin break(15); goto 15; end;
    15:if (accept(ans,15,a)) then goto 16;
       goto 99;
    16:if okay(a) then
       begin
           query:=true;
           return;
       end;
    99:query:=false
end;
```

Figure 3.11 Function QUERY

```
begin
    message(' EXECUTION BEGINS....',0);
    resatisfy:=false;
    sign:=';';
    while (sign=';') do
    begin
        if query(gg,1) then
        begin
            message(' yes',0);
            print;
            termin(term);
            readln(term,sign);
            close(term);
            resatisfy:=(sign=';');
            if resatisfy then
               message(' RESATISFYING GOAL....',0)
            else message(' EXECUTION ENDS....',0);
            continue;
        end;
        message(str(' no'),0);
        message(' EXECUTION ENDS....',0);
        halt;
    end;
end. (* main *)
```

Figure 3.12 Main Program

Then the global variable "resatisfy" is set to the "true". Otherwise execution ends. If "query" does have "true" value, after the execution, the procedure "print" prints the values of variables which are declared in the "query".

IV. IMPLEMENTATION AND TEST

A. SOME FEATURES OF PASCAL/VS AT NPS

Release 2.1 of Pascal/VS has several differences from "standard" Pascal. Most of the deviations are in the form of extensions to Pascal in those areas where Pascal does not have suitable facilities. We summarize some of them in Appendix A so that the interested user may understand the application programs given in Appendix B without having any surprise.

B. IMPLEMENTATION

This implementation involves mainly two distinct phases. The first phase is the compilation process (compiler or translator) and the second one is the executing process (executor). The Translator accepts source Prolog and translates it to Pascal source (object program) by including the necessary source and run-time routines. Then, the object program is compiled and executed under Pascal/VS system. All necessary files are handled automatically without requiring any user intervention. The main difference from a standard Prolog is that the user is asked to place his query as the last rule of the program. This rule must begin with the keyword "query".

The compiling process consists of three phases. These are:

1) Lexical Analyzing
2) Parsing
3) Translation

Compilation begins with a source Prolog file named "SOURCE PROLOG" which is created as a CMS file. (See Appendix C for a sample source program). The access to this file is sequential by the compiler. The token sequence is emitted by the lexical analyzer. If there is no rejected token, the parsing phase begins. The parser considers the context of each token and classifies groups of tokens such as variables, atoms or integers and also structures (rules, head or body clauses). For our purposes we introduce the main driver of the parsing process (see Figure 4.1) for the SPROLOG whose formal definition has been given in Chapter 3. The user may examine the other parts of the Parser by referring to the complete program which is given in Appendix B.

The product of parser and lexical analyzer are the tables described in Chapter 3 and also given in Appendix F, G, and H. The tables have two main jobs. First of all, the translator will use them for translation purposes. In fact, they are all parameters to be passed from user source program to object program. This makes explicit their second job. Namely, the executer embedded in the object code will use them during the execution.

```
function proc:boolean;
label a1;
begin
    if not(prefix) then
    begin
        result:=t(.tokenindex.).name;
        i:=t(.tokenindex.).linenum;
        message
        ('error... structure   expected.. '
        ||str(result),i);
        proc:=false;
        return;
    end;

    if point then
    begin
        proc:=true;
        return;
    end;
    reject;

    if not(iff) then
    begin
        result:=t(.tokenindex.).name;
        i:=t(.tokenindex.).linenum;
        message
        ('error... "." or ":-" expected.. '
        ||str(result),i);
        proc:=false;
        return;
    end;

a1: if not(structure) then
    begin
        result:=t(.tokenindex.).name;
        i:=t(.tokenindex.).linenum;
        message
        ('error... structure   expected.. '
        ||str(result),i);
        proc:=false;
        return;
    end;

    if point then
    begin
        proc:=true;
        return;
    end;
    reject;

    if comma then
       goto a1;

    proc:=false;
    result:=t(.tokenindex.).name;
    i:=t(.tokenindex.).linenum;
    message
    ('error... structure   expected.. '
    ||str(result),i);
end; (* proc *)
```

Figure 4.1   Main Driver for Parsing SPROLOG

As the last step, the translator translates the user source Prolog into Pascal. The mapping process between source and object program is given in Chapter 3. The assumptions and restrictions we have made before, make Pascal's "forward" declarations unnecessary. Also, the passing of integer pointers as parameters between the procedures prevents exhaustive variant record declarations. The probable recursive declarations made by the user in the source programs are detected in this phase by using the stack. Also, it is impossible to translate undefined procedures into Pascal. This process is handled by using the stack as well.

If there are no compiler detected errors, the Translator creates a Pascal source program (see Appendix C) which is called "USER PASCAL". During the creation phase the system library is used for the predefined procedures. After the creation of Pascal source code, the Pascal/VS compiler is called and "USER PASCAL" is compiled and executed. This is an interactive session. If the programmer is not content with an answer to his question, he can initiate backtracking himself by typing a semicolon when Prolog informs him of a solution.

C. TEST, EFFICIENCY CONSIDERATIONS AND SELF-CRITICISM

A sample program that has been compiled and translated into Pascal is given in the appendices. All these applications may be considered as relational database applications. The conjunction of many subgoals allows the user to define many queries.

This implementation does not make as efficient use of time and space as any commercial Prolog compiler or interpreter. The translation phase and compiling object code are all time consuming processes. Object code could be any assembly object code rather than Pascal, because Pascal/VS is also a slow compiler. On the other hand, it is apparent that a Prolog compiler spends a lot of its time backtracking. Backtracking is considered an unusual and expensive event in most language systems. Since in Prolog backtracking is the rule rather than exception, much of the challenge of Prolog implementation is the development of more efficient backtracking mechanisms. [Ref. 7]

It seems that the most important point of this work was not to write an efficient compiler. Rather our aim was to find a mapping system between Prolog and Pascal. But, this process also should be developed.

V. EPILOGUE

In this implementation we tried to translate a small subset of the programming language Prolog into Pascal. We discussed a mapping algorithm and we pointed out some difficulties.

In the literature there are many Prolog implementations. Many of them are interpreters (see Figure 5.1). For some implementations the reader may refer to references 8 and 9. Also, for the memory management of Prolog, see reference 10.

name	authors	implementation
PROLOG (Edinburgh Univ.)	F.M. Pereira F.C.N. Pereira D.H.D. Warren L. Byrd	MACRO (etc.) Dec Tops-10 -20
PROLOG (Marseille Univ.)	G. Battani H. Meloni	FORTRAN
PROLOG (IBM)	J.F. Sowa G. Roberts	VM/CMS
PROLOG/KR (Tokyo Univ.)	H. Nakashima	UTILISP

Figure 5.1 Prolog Systems

So let us review how one might set about constructing a compiler. Initially, the picture is just a black box with source programs as input and correctly translated object

programs as output. The first consideration is to decide how the output is related to the input. It is natural to examine the structure of the source language and to devise for each element of the language a rule for translating it into target language code. These rules form a specification of the compiler's function. The final and generally more laborious stage of compiler construction involves implementing procedures which efficiently carry out the translation process in accordance with the specification.

The SPROLOG implementation uses the primitive data structures, such as integer numbers, atomic constants and simple variables. List and tree types of data structures have not been considered. In the design phase we tried to give some idea for these structures. This requires efficient memory management processing. From this point, this thesis should be developed.

Backtracking should be considered as the most important fact in Prolog implementations. In particular, the existence of the long chain of variables during the execution phase, requires much more efficient compilation techniques.

We must sometimes take into account the way Prolog searches the database and what state of instantiation our goals will have in deciding the order in which to write the clauses of a Prolog program. The problem with introducing cuts is that we have to be even more certain of exactly how the rules of the program are to be used. For, whereas a cut

when a rule is used one way can be harmless or even beneficial, the very same cut can cause strange behaviour if the rule is suddenly used in another way. However, the cut operation would be introduced by defining a function to our implementation. But, we desired to give importance to relational database applications. For this reason, this operation is missing in this implementation.

Pascal has been chosen as an implementor language. The type checking and strong typing implies that careful design and planning should be considered in the compiler writing process. In particular, this language does not allow one to define twice names in the same context. Prolog does not restrict this. So, we renamed the user's procedure names when translating them. However, Ada does allow one to define procedures with same name (but with different number of parameters) in a given context. This language would provide much more features for this implementation.

As a conclusion we want to emphasize that the programming language Prolog itself also has more advantages than other existing conventional programming languages for writing a Prolog Compiler and also other compilers. Many of the advantages should be clear from the discussions that we have made so far. It is important to take into account, not just the compiler which is the product, but also the work which must go initially in designing and building it and into subsequently maintaining it.

To summarize, Prolog has the following advantages as a compiler-writing tool: less time and effort is required, there is less likelihood of error and the resulting implementation is easier to maintain and modify. Here is the last and most important sentence of this thesis: Prolog will be the programming language of the 20th Century.

APPENDIX A

SOME FEATURES OF PASCAL/VS

1) Separately compileable modules are supported with the SEGMENT definition.

2) "Internal static" data is supported by means of the "static" declarations.

3) "External static" data is supported by means of "def" and "ref" declarations.

4) Static and external data may be initialized at compile time by means of the "value" declarations.

5) Constant expressions are permitted wherever a constant is permitted except as the lower bound of a subrange type definition.

6) The keyword "range" may be prefixed to a subrange type definition to permit the lower value to be a constant expression.

7) A varying length character string is provided. It is called STRING. The maximum length of a STRING is 32367 characters.

8) The STRING operators and functions are CONCATENATE, LENGTH, STR, SUBSTR, DELETE, TRIM, LTRIM, COMPRESS and INDEX.

9) A new predefined type, STRINGPTR, has been added that permits the programmer to allocate strings with the NEW procedure whose maximum size is not defined until the invocation of NEW.

10) A new parameter passing mechanism is provided that allows strings to be passed into a procedure or function without requiring the programmer to specify the maximum size of the string on the formal parameter.

11) The MAIN directive permits the programmer to define a procedure that may be invoked from a non Pascal environment.

12) Files may be accessed based on relative record number (random access).

13) The tagfield in the variant part of a record may be anywhere within the fixed part of the record.

14) A parameter passing mechanism (const) has been defined which guarantees that the actual parameter is not modified yet does not require the copy overhead of a pass by value mechanism.

15) "leave", "continue" and "return" are new statements that permit a branching capability without using a "goto".

16) Labels may be either a numeric value or an identifier.

17) "case" statements may have a range notation on the component statements.

18) An "otherwise" clause is provided for the "case" statement.

19) The variant labels in records may be written with a range notation.

20) Constants may be of a structured type (namely arrays and records).

The other features which are not included here, are not directly related to our application. The concerned user may refer to Pascal/VS Manuals at NPGS.

APPENDIX B

TRANSLATOR FOR SPROLOG

```
program npro(input,output);
ccnst max=1500;
type trec=record
        linenum:integer;
        relnum:integer;
        name:alpha;
        ttype:integer;
        locality:integer;
      end;
     ttype=array(.1..max.) of trec;
    procrec=record
        rulenum:integer;
        relnum:integer;
        name:alpha;
        ptype:integer;
        relativity:integer;
        pointer1:integer;
        pointer2:integer;
        bbegin:integer;
        bend:integer;
        abegin:integer;
        aend:integer;
        yesno:integer;
        callee:integer;
        as:integer;
        ae:integer;
        now:integer;
        pom:integer;
    end;
    parrec=record
        rulenum:integer;
        relnum:integer;
        name:alpha;
        ptype:integer;
        locality:integer;
        pointer:integer;
        ntype:integer;
        nbind:integer;
        nmatch:bcclean;
    end;
var t:ttype;
    line,tokenindex,tbound,i,pend,pbegin:integer;
    query:boolean;
    date,time:alfa;result:alpha;
    lexerror,tokenerror:boolean;
    procfile,paramfile,listing,altfile:text;
    lib1,lib2,lib3,lib4,user:text;
    px,tx,ax,qq,ret:integer;
    proc:array(.1..max.) of procrec;
    par:array(.1..max.) of parrec;
    alt:array(.1..max.) of procrec;
```

```
procedure cms(const parmstr:string; var rc:integer);
    external;
procedure message(const msg:string;valint:integer);
var term:text;
begin
    termout(term);
    if (valint>0) then
    begin
       writeln(term,valint:3,str('.    ')||msg);
       writeln(listing,valint:3,str('.    ')||msg);
    end
    else if (valint=0) then
    begin
       writeln(term,msg);
       writeln(listing,msg);
    end
    else
    begin
       writeln(term,msg,(-valint));
       writeln(listing,msg,(-valint));
    end;
    close(term);
end;
function strlen(const instr:string):integer;
var chset:set of char; j,i:integer;
begin
    j:=0;
    chset:=(.'0'..'9','a'..'z'.);
    chset:=chset+not(chset);
    chset:=chset-(.' '.);
    for i:=1 to length(instr) do
    begin
        if (instr(.i.) in chset) then
           j:=succ(j);
    end;
    strlen:=j;
end;
```

```
procedure checktokens;
const maxtoken=17;
      legaltoken=16;
type
   rec= record
      res:alpha;
   end;
var hashtable:array(.1..max.) of integer;
    tokens:array(.1..maxtoken.) of alpha;
    totaltoken:integer;
    tokenfile:text; before:alpha;
    hashbound,j,reltoken,rule:integer;
    source:string(70);
    outfile:file of rec;
    pasfile:text;
procedure taketokens;
var
    taken:alpha;
    dummy:integer;
begin
    reset(tokenfile,'name=ptoken.input.a');
    while not(eof(tokenfile)) do
    begin
       readln(tokenfile,dummy,taken);
       taken:=ltrim(str(taken));
       tokens(.dummy.):=taken;
    end;
    close(tokenfile);
end;
function identifier:boolean;
var idset:set of char;
    i:integer;
begin
     idset:=(.'a'..'z','_'.);
     if (result(.1.) in idset ) then
     begin
        identifier:=true;
        return;
     end;
     identifier:=true;
     for i:=1 to strlen(str(result)) do
     begin
        if(not(result(.i.) in idset )) then
        begin
          identifier:=false;
          return;
        end;
     end;
end;
```

```
function atom:boolean;
var idset:set of char;
    i:integer;
begin
     idset:=(.'a'..'z'.);
     atom:=(result(.1.) in idset);
end;
function number:boolean;
var numset:set of char;
     i:integer;
begin
     numset:=(.'0'..'9'.);
     number:=true;
     for i:=1 to strlen(str(result)) do
     begin
         if (not(result(.i.) in numset )) then
         begin
             number:=false;
           return;
         end;
     end;
end;
```

```
procedure whichtcken(i:integer);
var j,ln:integer; tokenfound:boolean;
static hashindex:integer;
value hashindex:=0;
begin
    tokenfound:=false;
    for j:=1 to maxtoken do
    begin
        if(result=tokens(.j.)) then
        begin
            hashindex:=succ(hashindex);
            hashtable(.hashindex.):=j;
            if (j>legaltoken) then
            begin
                ln:=t(.i.).linenum;
                message
                ('erroneous token: '||str(result),ln);
                tckenerror:=true;
            end;
            tokenfound:=true;
            leave;
       end;
    end;
       if (not(tckenfound)) then
       begin
           if identifier then
           begin
               hashindex:=succ(hashindex);
               hashtable(.hashindex.):=succ(maxtoken);
               tckenfound:=true;
           end;
       end;
       if (not(tckenfound)) then
       begin
           if number then
           begin
               hashindex:=succ(hashindex);
               hashtable(.hashindex.)
               :=succ(succ(maxtoken));
               tokenfound:=true;
           end;
       end;
       if (not(tokenfound)) then
       begin
           if atcm then
           begin
               hashindex:=succ(hashindex);
               hashtable(.hashindex.)
               :=succ(succ(maxtoken));
               tckenfound:=true;
           end;
       end;
```

```
        if (not(tokenfound)) then
        begin
            tokenerror:=true;
            ln:=t(.i.).linenum;
            message
            ('error... token=====> '||str(result),ln);
        end;
        hashbound:=hashindex;
end;
```

```
procedure putfile(i:integer);
begin
    writeln(rule:4,reltoken:4,'    ',result);
    outfile@.res:=result;
    put(outfile);
    t(.i.).linenum:=rule;
    t(.i.).relnum:=reltoken;
end;
procedure rejecttoken;
var i,j:integer; b:ttype; tf:boolean;
    opset:set of char;
begin
    opset:=(.'+','-','*','/'.);
    message(' ',0);
    for i:=1 to (tbound-1) do
    begin
        tf:=(t(.i.).name=':')
           (t(.i+1.).name=':-');
        if tf then
           t(.i.).name:=' ';
    end;
    j:=0;
    for i:=1 to tbound do
    begin
        tf:=(not(t(.i.).name=' '));
        if tf then
        begin
           j:=succ(j);
           b(.j.).name:=t(.i.).name;
        end;
    end;
    tbound:=j;
    t:=b;
    if (t(.tbound.).name <> '.') then
    begin
        message
        ('warning.. no eof?  "."  assumed'
        ,t(.tbound.).linenum);
        t(.tbound+1.).name:='.';
        tbound:=succ(tbound);
    end;
    rule:=1; reltoken:=1;
    for i:=1 to tbound do
    begin
       result:=t(.i.).name;
       with t(.i.) do
       begin
           if (identifier or
                atom     or
                number   or
               (result(.1.) in opset))
           then locality:=0
           else locality:=-1;
```

```
        end;
        putfile(i);
        if (result='.') then rule:=succ(rule);
        if (result='.') then reltoken:=0;
        reltoken:=succ(reltoken);
        whichtoken(i);
     end;
  end;
```

```
procedure puttoken;
begin
    before:=result;
    totaltoken:=succ(totaltoken);
    tokenindex:=succ(tokenindex);
    t(.tokenindex.).name:=result;
    tbound:=tokenindex;
end; (*put token *)
procedure tokenfound;
var i:integer;
static proc:integer;
value proc:=0;
begin
    taketokens;
    totaltoken:=0;
    reset(pasfile,'name=source.prolog.a');
    rewrite(outfile);
    line:=0;
    while not(eof(pasfile)) do
    begin
        readln(pasfile,source);
        if (source<>str(' '))
           then line:=succ(line);
        if (source<>str(' '))
           then message(source,line);
        source:=compress(source);
        j:=length(source);
        i:=1;
        while (i<=j) do
        begin
            token(i,source,result);
            if (result='.') then
                proc:=succ(proc);
            if (result<>' ')then
            begin
                if (result='-') then
                    if(before=':') then
                        result:=str(':-');
                puttoken;
           end;
        end;
    end;
    rejecttoken;
end; (* tokenfound*)
begin
    rewrite(listing);
    before:=str('ā');
    tokenerror:=false;
    tokenfound;
end; (* checktokens *)
```

```
procedure lexicalanalyzer;
type xtype=array(.1..7.) of integer;
     ptype=record
        name:alpha;
        numb:integer;
     end;
var prec:array(.1..12.) of ptype;
    a:array(.1..80,1..7.) of integer;
    global,null:boolean;  ttoken:alpha;
procedure  takeexp;
var expfile:text; i,j:integer;
begin
    reset(expfile,'name=exp.input.a1');
    i:=0;
    while not(eof(expfile)) do
    begin
        i:=succ(i);
        for j:=1 to 7 do
        begin
            read(expfile,a(.i,j.));
        end;
        readln(expfile);
    end;
end; (*takeexp*)
procedure give;
begin
    prec(.1.).name:='+';
    prec(.1.).numb:=5;
    prec(.2.).name:='-';
    prec(.2.).numb:=5;
    prec(.3.).name:='*';
    prec(.3.).numb:=5;
    prec(.4.).name:='/';
    prec(.4.).numb:=5;
    prec(.5.).name:='=';
    prec(.5.).numb:=6;
    prec(.6.).name:='<';
    prec(.6.).numb:=7;
    prec(.7.).name:='>';
    prec(.7.).numb:=7;
    prec(.8.).name:='<>';
    prec(.8.).numb:=7;
    prec(.9.).name:='<=';
    prec(.9.).numb:=7;
    prec(.10.).name:='>=';
    prec(.10.).numb:=7;
    prec(.11.).name:='is';
    prec(.11.).numb:=4;
    prec(.12.).name:='is';
    prec(.12.).numb:=4;
end;
```

```
function taketoken:alpha;
begin
    if global then
       tokenindex:=succ(tokenindex);
    null:=(tokenindex>pend);
    assert not(null);
    if not(null) then
       taketoken:=t(.tokenindex.).name
    else
       taketoken:='@';
end; (* taketoken *)
procedure reject;
begin
    tokenindex:=pred(tokenindex);
end; (* reject *)
function left:boolean;
begin
    left:=false;
    if not(null) then
       left:=(taketoken='(');
end;

function right:boolean;
begin
    right:=false;
    if not(null) then
       right:=(taketoken=')');
end;
function comma:boolean;
begin
    comma:=false;
    if not(null) then
       comma:=(taketoken=',');
end;
function variable:boolean;
var idset:set of char;
    i:integer;
begin
     variable:=false;
     result:=taketoken;
     idset:=(.'a'..'z','_'.);
     if (result(.1.) in idset ) then
     begin
         variable:=true;
         return;
     end;
end;
```

```
function atom:boolean;
var idset:set of char;
    i:integer;
begin
     result:=taketoken;
     idset:=(.'a'..'z'.);
     atom:=(result(.1.) in idset);
end;
function number:boolean;
var numset:set of char;
     i:integer;
begin
     numset:=(.'0'..'9'.);
     number:=true;
     result:=taketoken;
     for i:=1 to strlen(str(result)) do
     begin
         if (not(result(.i.) in numset )) then
         begin
             number:=false;
           return;
         end;
     end;
end;
function varorconst:boolean;
begin
    if variable then
    begin
        t(.tokenindex.).ttype:=1;
        varorconst:=true;
        return;
    end
    else reject;
    if number then
    begin
        t(.tokenindex.).ttype:=2;
        varorconst:=true;
        return;
    end
    else reject;
    if atom then
    begin
        t(.tokenindex.).ttype:=3;
        varorconst:=true;
        return;
    end;
    varorconst:=false;
end;
```

```
function iff:boolean;
begin
    iff:=false;
    if not(null) then
       iff:=(taketoken=':-');
end;
function procname:boolean;
begin
    procname:=atcm;
end;
function point:boolean;
begin
    point:=false;
    if not(null) then
       point:=(taketoken='.');
end;
function prefix:boolean;
label a1;
var local:integer;
begin
    local:=0;
    if not(procname) then
    begin
        prefix:=false;
        return;
    end;
    if not(left) then
    begin
        reject;
        local:=0;
        t(.tokenindex.).ttype:=4;
        prefix:=true;
        return;
    end;
        t(.tokenindex-1.).ttype:=5;
a1: if not(varorconst) then
    begin
        prefix:=false;
        return;
    end;
    local:=succ(local);
    t(.tokenindex.).locality:=local;
    if right then
    begin
        prefix:=true;
        return;
    end;
    reject;
    if comma then
       goto a1;
    prefix:=false;
end; (* prefix *)
```

```
function expressicn(inex:xtype):boolean;
var cex,f:xtype;
procedure convert;
var i,j,ix,cindex:integer;
begin
    for i:=1 to 7 do
        cex(.i.):=0;
    for i:=1 to 7 do
    begin
        if (inex(.i.)=0) then continue;
        for j:=1 to 12 do
        begin
            ix:=inex(.i.);
            if (t(.ix.).name = prec(.j.).name) then
            begin
                cex(.i.):=prec(.j.).numb;
                leave;
            end;
        end;
    end;
    global:=false;
    for i:=1 to 7 do
    begin
        if(inex(.i.)=0) then continue;
        if(cex(.i.)>0) then continue;
        tokenindex:=inex(.i.);
        if variable then
        begin
            t(.tokenindex.).ttype:=1;
            cex(.i.):=1;
            continue;
        end;
        if number then
        begin
            t(.tokenindex.).ttype:=2;
            cex(.i.):=2;
            continue;
        end;
        if atom then
        begin
            t(.tokenindex.).ttype:=3;
            cex(.i.):=3;
            continue;
        end;
    end;
    global:=true;
end; (* convert *)
```

```
function check:bcclean;
var i:integer; res:boolean;
begin
    res:=true;
    for i:=1 to 7 do
    begin
        res:=(cex(.i.)=f(.i.)) and res;
        if not(res) then leave;
    end;
    check:=res;
end;
function send:boclean;
var i,j:integer;
begin
    send:=false;
    for i:=1 to 80 do
    begin
        for j:=1 to 7 do
            f(.j.):=a(.i,j.);
        if check then
        begin
            send:=true;
            return;
        end;
    end;
end;
begin   (* expression *)
    convert;
    expression:=send;
end; (* expressicn *)
```

```
function infix:boolean;
label a1;
type rc=record
       name:alpha;
       ind:integer;
      end;
var i,cindex,middle,len,j,tm:integer;
    ex:xtype;
    legal,legal1,legal2:boolean;
    tok:array(.1..8.) of rc;
begin
    cindex:=tokenindex;
    i:=0;
    repeat
         i:=succ(i);if (i>8) then leave;
         ttoken:=taketoken;
         tok(.i.).name:=ttoken;
         tok(.i.).ind:=tokenindex;
    until (ttoken='.') or (ttoken=',');
    len:=pred(tokenindex-cindex);
    tm:=0;
    for i:=1 to len do
    begin
      for j:=5 to 12   do
        if (tok(.i.).name=prec(.j.).name) then
        begin
            tm:=i;
            goto a1;
        end;
    end;
a1:if (tm=0) then
    begin
        tokenindex:=cindex;
        infix:=false;
        return;
    end;
    for i:=1 to 7 do
        ex(.i.):=0;
    j:=3;
    for i:=1 to tm do
    begin
        if((tm-i)<1) then leave;
        if(j<1) then leave;
        ex(.j.):=tok(.tm-i.).ind;
        j:=pred(j);
    end;
    j:=4;
    for i:=tm to len do
    begin
        if(j>7) then leave;
        ex(.j.):=tok(.i.).ind;
        j:=succ(j);
    end;
```

```
      j:=0;
      if expressicn(ex) then
      begin
          t(.ex(.4.).).ttype:=6;
          for i:=1 to 7 do
          begin
              if (ex(.i.)=0) then continue;
              j:=succ(j);
              t(.ex(.i.).).locality:=j;
              if (t(.ex(.i.).).name='+') then
                  t(.ex(.i.).).ttype:=7;
              if (t(.ex(.i.).).name='-') then
                  t(.ex(.i.).).ttype:=8;
              if (t(.ex(.i.).).name='*') then
                  t(.ex(.i.).).ttype:=9;
              if (t(.ex(.i.).).name='/') then
                  t(.ex(.i.).).ttype:=10;
          end;
          infix:=true;
          return;
      end;
     infix:=false;
     tokenindex:=cindex;
 end;
```

```
function structure:boolean;
begin
    if infix then
       structure:=true
    else structure:=prefix;
end;
function proc:boolean;
label a1;
begin
    if not(prefix) then
    begin
        result:=t(.tokenindex.).name;
        i:=t(.tokenindex.).linenum;
        message
        ('error... structure   expected.. '
        ||str(result),i);
        proc:=false;
        return;
    end;
    if point then
    begin
        proc:=true;
        return;
    end;
    reject;
    if not(iff) then
    begin
        result:=t(.tokenindex.).name;
        i:=t(.tokenindex.).linenum;
        message
        ('error... "." or ":-" expected.. '
        ||str(result),i);
        proc:=false;
        return;
    end;
a1: if not(structure) then
    begin
        result:=t(.tokenindex.).name;
        i:=t(.tokenindex.).linenum;
        message
        ('error... structure   expected.. '
        ||str(result),i);
        proc:=false;
        return;
    end;
    if point then
    begin
        proc:=true;
        return;
    end;
    reject;
```

```
     if comma then
        goto a1;
     proc:=false;
     result:=t(.tckenindex.).name;
     i:=t(.tokenindex.).linenum;
     message
     ('error... structure   expected.. '
     ||str(result),i);
  end; (* proc *)
```

```
begin (* lexical analyzer *)
    lexerror:=false;
    global:=true;
    takeexp; give;
    null:=false; tokenindex:=0;
    i:=0; pbegin:=1;
    while (i<=tbound) do
    begin
        repeat
            i:=succ(i);
            if(i>tbound) then leave;
        until (t(.i.).name = '.');
        if(i>tbound) then leave;
        pend:=i;
        if not(prcc) then
        begin
            lexerror:=true;
            i:=pend;
            tokenindex:=pend;
        end;
        pbegin:=succ(i);
    end;
end;  (* lexical analyzer *)
procedure changea;
var i,q,r,bb,be:integer;
    dummy:alpha;
begin
    query:=false;
    q:=0;
    r:=0;
    for i:=1 to px do
    begin
        dummy:=' ';
        dummy:=trim(str(prcc(.i.).name));
        if(dummy='a') then
            proc(.i.).name:='$';
        if(dummy='query') then
        begin
           q:=succ(q);
           r:=i;
        end;
    end;
    if(q=1) then
    begin
      query:=true;
      bb:=proc(.r.).bbegin;
      be:=proc(.r.).bend;
      if(bb=0) then
      begin
          query:=false;
          message
 (' error..."query" must be defined as a rule',r);
```

```
        end;
        return;
     end;
     if(q=0) then
     begin
        query:=false;
        message
 (' error...there must be a "query" procedure',r);
        return;
     end;
     if(q>1) then
     begin
        query:=false;
        message
 (' error...more than one "query" procedures',r);
     end;
end; (*changea*)
```

```
procedure createarrays;
var a,b,i,j,count,before:integer;
procedure alternatives;
var i,ab,ae:integer;
    passname:alpha;
    a,b:integer;
procedure putalternate
          (passname:alpha; var ab,ae:integer);
var i,j:integer;
static x:integer;
value x:=1;
begin
    ab:=x;
    for i:=1 to px do
    begin
        if(proc(.i.).relativity<>0) then continue;
        if(proc(.i.).name<>passname) then continue;
        alt(.x.):=proc(.i.);
        alt(.x.).aend:=i;
        alt(.x.).abegin:=proc(.i.).rulenum;
        ae:=x;
        x:=succ(x);
    end;
    if(ab<>0) then
    begin
        if((ab-ae)=0) then
        begin
            ab:=0;
            ae:=0;
            x:=pred(x);
        end;
    end;
    ax:=pred(x);
end;
procedure putthenumber
          (passname:alpha; ab,ae,i:integer);
var j:integer;
begin
    for j:=i to px do
    begin
        if(proc(.j.).relativity=0) then continue;
        if(proc(.j.).name<>passname) then continue;
        proc(.j.).abegin:=ab;
        proc(.j.).aend:=ae;
    end;
end; (* putthenumber *)
```

```
procedure call;
var i,ij,j:integer;
begin
    for i:=1 to px do
    begin
        proc(.i.).yesno:=0;
        proc(.i.).callee:=0;
    end;
    for i:=1 to ax do
    begin
        ij:=alt(.i.).abegin;
        proc(.ij.).yesno:=1;
    end;
    for i:=1 to px do
    begin
        if(proc(.i.).yesno=1) then continue;
        if(proc(.i.).abegin>0) then continue;
        if(proc(.i.).ptype=6) then continue;
        if(proc(.i.).callee>0) then continue;
        if(proc(.i.).relativity>0) then continue;
        for j:=1 to px do
        begin
            if(proc(.j.).yesno=1) then continue;
            if(proc(.j.).abegin>0) then continue;
            if(proc(.j.).ptype=6) then continue;
            if(proc(.j.).callee>0) then continue;
            if(proc(.j.).relativity=0) then continue;
            if(proc(.i.).name<>proc(.j.).name) then
            continue;
            proc(.j.).callee:=i;
        end;
    end;
end; (* call *)
```

```
begin
    for i:=1 to px do
    begin
        proc(.i.).abegin:=0;
        proc(.i.).aend:=0;
        alt(.i.).abegin:=0;
        alt(.i.).aend:=0;
    end;
    for i:=1 to px do
    begin
        if(proc(.i.).relativity=0) then continue;
        if(proc(.i.).ptype=6) then continue;
        if(proc(.i.).abegin<>0) then continue;
        passname:=proc(.i.).name;
        putalternate(passname,ab,ae);
        putthenumber(passname,ab,ae,i);
    end;
    i:=0;
    for a:=240 tc 249 do
       for b:=240 to 249 do
       begin
           if((a=240) and (b=240)) then continue;
           i:=succ(i);
           if(i>ax) then leave;
           if(a<>240) then
           alt(.i.).name:=trim(str(alt(.i.).name))||
                             str(char(a))||str(chr(b))
           else
           alt(.i.).name:=trim(str(alt(.i.).name))||
                             str(chr(b));
       end;
       call;
end; (* alternatives *)
procedure procdo(t:trec);
begin
    px:=succ(px);
    proc(.px.).rulenum:=t.linenum;
    proc(.px.).relnum:=t.relnum;
    proc(.px.).name:=t.name;
    proc(.px.).ptype:=t.ttype;
end;
```

```
procedure pardo(t:trec);
begin
    tx:=succ(tx);
    par(.tx.).rulenum:=t.linenum;
    par(.tx.).relnum:=t.relnum;
    par(.tx.).name:=t.name;
    par(.tx.).ptype:=t.ttype;
    par(.tx.).locality:=t.locality;
end;
procedure beginend;
label a1,a2;
var i,j:integer;
begin
    for i:=1 to px do
    begin
        proc(.i.).bbegin:=0;
        proc(.i.).bend:=0;
    end;
    i:=0; j:=0;
a1: repeat
         i:=succ(i);
         if(i>px) then return;
     until (proc(.i.).relativity=1);
     j:=pred(i);
     proc(.j.).bbegin:=i;
    repeat
         i:=succ(i);
         if(i>px) then goto a2;
     until (proc(.i.).relativity=0);
     proc(.j.).bend:=pred(i);
     goto a1;
a2:  proc(.j.).bend:=px;
end; (* beginend *)
```

```
procedure putfiles;
var i:integer;
begin
    rewrite(procfile);
    rewrite(paramfile);
    rewrite(altfile);
    for i:=1 to tx do
    begin
        write(paramfile,i:3,'.':1);
        write(paramfile,par(.i.).rulenum:4);
        write(paramfile,par(.i.).relnum:4);
        write(paramfile,' ':4);
        write(paramfile,par(.i.).name:12);
        write(paramfile,par(.i.).ptype:4);
        write(paramfile,par(.i.).locality:4);
        write(paramfile,par(.i.).pointer:4);
        write(paramfile,par(.i.).ntype:4);
        write(paramfile,par(.i.).nbind:4);
        writeln(paramfile);
    end;
    for i:=1 to px do
    begin
        write(procfile,i:3,'.':1);
        write(procfile,proc(.i.).rulenum:4);
        write(procfile,proc(.i.).relnum:4);
        write(procfile,' ':4);
        write(procfile,proc(.i.).name:12);
        write(procfile,proc(.i.).ptype:4);
        write(procfile,proc(.i.).relativity:4);
        write(procfile,proc(.i.).pointer1:4);
        write(procfile,proc(.i.).pointer2:4);
        write(procfile,proc(.i.).bbegin:4);
        write(procfile,proc(.i.).bend:4);
        write(procfile,proc(.i.).abegin:4);
        write(procfile,proc(.i.).aend:4);
        write(procfile,proc(.i.).yesno:4);
        write(procfile,proc(.i.).callee:4);
        write(procfile,proc(.i.).as:3);
        write(procfile,proc(.i.).ae:3);
        write(procfile,proc(.i.).now:3);
        write(procfile,proc(.i.).pom:3);
        writeln(procfile);
    end;
    for i:=1 to ax do
    begin
        write(altfile,i:3,'.':1);
        write(altfile,alt(.i.).rulenum:4);
        write(altfile,alt(.i.).relnum:4);
        write(altfile,' ':4);
        write(altfile,alt(.i.).name:12);
        write(altfile,alt(.i.).ptype:4);
        write(altfile,alt(.i.).relativity:4);
```

```
        write(altfile,alt(.i.).pointer1:4);
        write(altfile,alt(.i.).pointer2:4);
        write(altfile,alt(.i.).bbegin:4);
        write(altfile,alt(.i.).bend:4);
        write(altfile,alt(.i.).abegin:4);
        write(altfile,alt(.i.).aend:4);
        writeln(altfile);
    end;
end; (*putfiles *)
```

```
procedure cnow;
var i,j,k:integer;
    pan1,pan2:alpha;
begin
    for i:=1 to px do
    begin
        proc(.i.).as:=proc(.i.).abegin;
        proc(.i.).ae:=proc(.i.).aend;
        proc(.i.).now:=proc(.i.).abegin;
        if(proc(.i.).as<>0) then continue;
        proc(.i.).as:=1;
        proc(.i.).ae:=1;
        proc(.i.).now:=1;
    end;
    for i:=1 to px do
        proc(.i.).pom:=0;
    k:=1;
    proc(.1.).pom:=1;
    for i:=1 to px do
    begin
        if(proc(.i.).pom=0) then
         begin
             k:=succ(k);
             proc(.i.).pom:=k;
         end;
        pan1:=proc(.i.).name;
        for j:=i+1 to px do
        begin
            pan2:=proc(.j.).name;
            if(pan1=pan2) then
               proc(.j.).pom:=proc(.i.).pom;
        end;
    end;
end; (*cnow*)
```

```
procedure genbind;
var i,j:integer;
begin
    for i:=1 to tx do
    begin
        par(.i.).nmatch:=false;
        if(par(.i.).ptype=1) then
        begin
            par(.i.).ntype:=0;
            par(.i.).nbind:=0;
            continue;
        end;
        par(.i.).ntype:=par(.i.).ptype;
        par(.i.).nbind:=i;
    end;
    for i:=1 to tx do
    begin
        if(par(.i.).ntype=0) then continue;
        for j:=1 to tx do
        begin
            if(par(.i.).name<>par(.j.).name)
            then continue;
            par(.j.).ntype:=par(.i.).ntype;
            par(.j.).nbind:=par(.i.).nbind;
        end;
    end;
end; (*genbind*)
```

```
begin
    px:=0; tx:=0;
    for i:=1 to tbound do
    begin
        case t(.i.).ttype of
        4,5,6:        procdo(t(.i.));
        1,2,3,7,8,9,10:pardo(t(.i.));
        otherwise tbound:=tbound;
        end;
    end;
    j:=1;
    for i:=1 to px do
    begin
        if  (j>tx) then leave;
        if  (proc(.i.).ptype=4) then
        begin
            proc(.i.).pointer1:=0;
            proc(.i.).pointer2:=0;
            continue;
        end;
        proc(.i.).pointer1:=j;
        par(.j.).pointer:=i;
        repeat
             j:=succ(j);
             if  (j>tx) then leave;
             par(.j.).pointer:=i;
        until
(par(.j.).locality<=par(.j-1.).locality);
        proc(.i.).pointer2:=pred(j);
    end;(*for*)
    before:=proc(.1.).rulenum;
    count:=0;
    for i:=1 to px do
    begin
        if(proc(.i.).rulenum=before) then
        begin
            proc(.i.).relativity:=count;
            count:=succ(count);
            continue;
        end;
        before:=proc(.i.).rulenum;
        count:=1;
    end;
    beginend;
    changea;
    alternatives;
    cnow;
    genbind;
    putfiles;
end; (* createarray *)
```

```
function defined:boolean;
var i,j,k,pm,bb,be:integer;  deff:boolean;
    nom:alpha;
begin
    defined:=true;
    for i:=1 to px do
     begin
         if(proc(.i.).relativity=0)
           then continue;
         if(proc(.i.).ptype=6) then continue;
         deff:=false;
         for j:=1 to px do
          begin
              if(proc(.j.).relativity<>0)
                then continue;
              if(proc(.j.).name=proc(.i.).name) then
              begin
                  deff:=true;
                  leave;
              end;
          end;
          if (not(deff)) then
           begin
              nom:=ltrim(trim(str(proc(.i.).name)));
              if(nom='$') then
                nom:='a';
              message
              ('undefined procedure  '||str(nom),
              proc(.i.).rulenum);
              defined:=false;
              return;
           end;
     end;
     for i:=1 to px do
     begin
         if(proc(.i.).ptype=6) then continue;
         if(proc(.i.).relativity>0) then continue;
         if(proc(.i.).bbegin=0) then continue;
         bb:=proc(.i.).bbegin;
         be:=proc(.i.).bend;
         for j:=bb to be do
         begin
             pm:=proc(.j.).pom;
             for k:=i to px do
             begin
                 if(proc(.k.).relativity>0)
                   then continue;
                 if(proc(.k.).pom<>pm) then continue;
                 nom:=ltrim(trim(str(proc(.k.).name)));
                 if(nom='$') then
                    nom:='a';
                    message
                    ('undefined procedure'||str(nom),
                    proc(.k.).rulenum);
                 defined:=false;
                 return;
             end;
         end;
    end;
end; (* defined *)
```

```
function recursive:boolean;
var i,j,k,l,v0,v1,v2,vb,ve,vb1,ve1:integer;
    xname,nom:alpha
begin
    recursive:=false;
    for i:=1 to px do
    begin
        if(proc(.i.).bbegin=0) then continue;
        xname:=proc(.i.).name;
        for j:=proc(.i.).bbegin to proc(.i.).bend do
        begin
            if(proc(.j.).name<>xname) then continue;
            recursive:=true;
            nom:=ltrim(trim(str(xname)));
            if(nom='$') then nom:='a';
            message
            ('recursive is not allowed'||str(nom),
            proc(.i.).rulenum);
            return;
        end;
    end;
    for i:=1 to px do
    begin
       if((proc(.i.).relativity=0) and
         (proc(.i.).bbegin>0)) then
        begin
            v0:=proc(.i.).pom;
            vb:=proc(.i.).bbegin;
            ve:=proc(.i.).bend;
            for j:=vb to ve do
            begin
                v1:=proc(.j.).pom;
                for k:=1 to px do
                begin
                   if(proc(.k.).relativity>0)
                     then continue;
                   if(proc(.k.).bbegin=0)
                     then continue;
                   if(proc(.k.).pom<>v1)
                     then continue;
                   vb1:=proc(.k.).bbegin;
                   ve1:=proc(.k.).bend;
                   for l:=vb1 to ve1 do
                   begin
                       v2:=proc(.l.).pom;
                       if(v2=v0) then
                       begin
                           recursive:=true;
                           xname:=proc(.l.).name;
                           nom:=ltrim(trim(str(xname)));
                           if(nom='$') then nom:=a;
                           message
                           ('recursive     is     not
allowed'
                           ||str(nom),
                           proc(.l.).rulenum);
                           return;
                       end; end; end;
                   end; end; end;
 end; (* recursive *)
```

```
procedure createpascal;
var
    ext:array(.1..255.) of alpha;
    p1,p2,p3,i,t:integer;pan:alpha;
function nl(n:integer):integer;
var i:integer;
begin
    nl:=2;
    i:=n div 10;
    if (i=0) then nl:=1;
end;
procedure takelib0;
var line:string(72);i:integer;
    pempty,aempty,tempty:boolean;
    p,a,t:string(6);
begin
    line:='program user(input,output);';
    writeln(user,line);
    line:='const';
    writeln(user,line);
    pempty:=(px=0);
    aempty:=(ax=0);
    tempty:=(tx=0);
    if pempty then px:=2;
    if aempty then ax:=2;
    if tempty then tx:=2;
    p:=str('false');
    a:=str('false');
    t:=str('false');
    if pempty then p:=str('true');
    if aempty then a:=str('true');
    if tempty then t:=str('true');
    writeln(user,'    pempty=',p:strlen(p),';');
    writeln(user,'    aempty=',a:strlen(a),';');
    writeln(user,'    tempty=',t:strlen(t),';');
    writeln(user,'    px=',px:nl(px),';');
    writeln(user,'    ax=',ax:nl(ax),';');
    writeln(user,'    tx=',tx:nl(tx),';');
    writeln(user,'    qq=',qq:nl(qq),';');
end;
procedure takelib1;
var line:string(72);
begin
    reset(lib1,'name=lib1.pascal.a1');
    while not eof(lib1) do
    begin
        readln(lib1,line);
        writeln(user,line);
    end;
    close(lib1);
end; (* takelib1*)
```

```
procedure takelib2;
var line:string(72);
begin
    reset(lib2,'name=lib2.pascal.a1');
    while not eof(lib2) do
    begin
        readln(lib2,line);
        writeln(user,line);
    end;
    close(lib2);
end; (* takelib2*)
procedure takelib3;
var line:string(72);
begin
    reset(lib3,'name=lib3.pascal.a1');
    while not eof(lib3) do
    begin
        readln(lib3,line);
        writeln(user,line);
    end;
    close(lib3);
end; (* takelib3*)
procedure takelib4;
var line:string(72);
begin
    reset(lib4,'name=lib4.pascal.a1');
    while not eof(lib4) do
    begin
        readln(lib4,line);
        writeln(user,line);
    end;
    close(lib4);
end; (* takelib4*)
```

```
function change(var p:alpha):alpha;
begin
    change:=p;
    if(p='<') then change:='lessthan';
    if(p='>') then change:='greater';
    if(p='<=') then change:='lessequal';
    if(p='>=') then change:='greatequal';
    if(p='<>') then change:='notequal';
    if(p='=') then change:='equal';
end;
function exist(p:alpha):boolean;
var i:integer;
begin
    exist:=false;
    for i:=1 to t do
        if(p=ext(.i.)) then exist:=true;
end;
procedure createfun(fname:alpha);
type l1=record
       a:string(9);
       n:alpha;
       o:string(22);
     end;
     l2=string(70);
     l3=record
       b:string(4);
       n:alpha;
       o:string(13);
     end;
var f1:l1;f2:l2;f3:l3;f4:l2;
begin
    if(exist(fname)) then return;
    f1.a:='function ';
    f1.n:=fname;
    f1.o:='(a,i:integer):boolean;';
    f2:='begin';
    f3.b:='    ';
    f3.n:=fname;
    f3.o:=':=match(a,i);';
    f4:='end;';
    with f1 do
```

```
        begin
           writeln(lib4,a:9,n:strlen(str(n)),o:22);
        end;
        writeln(lib4,f2);
        with f3 do
        begin
           writeln(lib4,b:4,n:strlen(str(n)),o:13);
        end;
        writeln(lib4,f4);
        t:=succ(t);
        ext(.t.):=fname;
    end;
```

```
procedure altbody(fname:alpha;abe,abn:integer);
type
   line=string(72);
   line1=record
     a:string(9);
     b:alpha;
     c:string(22);
   end;
   line4=record
     a:string(4);
     b:integer;
     c:char;
     d:alpha;
     e:string(2);
     f:alpha;
     g:alpha;
  end;
var l1:line1;l2,l3,l5,l6:line;l4:line4;i:integer;
begin
    if(exist(fname)) then return;
    t:=succ(t);
    ext(.t.):=fname;
    l1.a:='function ';
    l1.b:=fname;
    l1.c:='(a,i:integer):boolean;';
    l2:='begin';
    l3:='    case i of';
    write(lib4,l1.a);
    write(lib4,l1.b:strlen(str(l1.b)));
    writeln(lib4,l1.c);
    writeln(lib4,l2);writeln(lib4,l3);
    with l4 do
    begin
        a:='    ';
        c:=':';
        d:=fname;
        e:=':=';
        g:='(a,i);';
    end;
    for i:=abe to abn do
    begin
        with l4 do
        begin
            b:=i;
            f:=alt(.i.).name;
            t:=succ(t);
            ext(.t.):=f;
         write(lib4,a:4,b:nl(b),c:1,d:strlen(str(d)));
            writeln(lib4,e:2,f:strlen(str(f)),g:6);
       end;
    end;
    l5:='    end;';writeln(lib4,l5);
    l6:='end;';writeln(lib4,l6);
end; (*altbody*)
```

```
procedure createrule(fname:alpha;var a,b:integer);
type
   line=string(70);
   lrec=record
      two:integer;
      del:char;
   end;
   lab=record
     dec:string(6);
     num:array(.1..20.) of lrec;
     del:char;
   end;
   line1=record
       a:string(9);
       n:alpha;
       o:string(22);
     end;
   line3=record
       a:string(34);
       b:integer;
       c:string(8);
       d:integer;
       e:string(6);
     end;
   line6=record
       no:string(4);
       num1:integer;
       cond:alpha;
       name:alpha;
       comma1:char;
       num2:integer;
       comma2:char;
       a:char;
       other:string(13);
       num3:integer;
       del:char;
   end;
   line9=record
       f:string(23);
       num:integer;
       other:string(16);
   end;
   line10=record
        other:alpha;
        num:integer;
        f:string(2);
   end;
   line12=record
        b:string(4);
        n:integer;
        o:string(16);
   end;
```

```
    line14=record
         a:string(11);
         b:alpha;
         c:string(7);
    end;
    line17=record
         a:string(7);
         b:alpha;
         c:string(7);
    end;
var l,l4,l7,l13,l15,l16,l18:line; l3:line3;
     l2:lab;l6,l8:line6;l9:line9;
     l10,l11:line10;l12:line12;
     l1:line1; l14:line14; l17:line17;
     i,aa,li:integer;
```

```
procedure line15;
var i:integer;
begin
    l1.a:='function ';
    l1.n:=fname;
    l1.o:='(a,i:integer):boolean;';
    l2.dec:='label ';
    for i:=1 to 20 do
    begin
        l2.num(.i.).two:=0;
        l2.num(.i.).del:=',';
    end;
    l2.del:=';';
    with l3 do
    begin
        a:='    if resatisfy then begin break(';
        b:=0;
        c:='); goto ';
        d:=0;
        e:='; end;';
    end;
    l4:='begin';
end;
procedure line67;
begin
    l6.no:='     ';
    l6.num1:=0;
    l6.cond:=':if (accept(';
    l6.name:='        ';
    l6.comma1:=',';
    l6.num2:=0;
    l6.comma2:=',';
    l6.a:='a';
    l6.other:=')) then goto ';
    l6.num3:=0;
    l6.del:=';';
    l7:='        goto 99;';
end;
procedure line811;
begin
    l8.no:='     ';
    l8.num1:=0;
    l8.cond:=':if (accept(';
    l8.name:='        ';
    l8.comma1:=',';
    l8.num2:=0;
    l8.comma2:=',';
    l8.a:='a';
    l8.other:=')) then goto ';
    l8.num3:=0;
    l8.del:=';';
    l9.f:='        if not(possible(';
```

```
        l9.num:=0;
        l9.other:=')) then goto 99;';
        l10.other:='        break(';
        l10.num:=0;
        l10.f:=');';
        l11.other:='        goto ';
        l11.num:=0;
        l11.f:=';';
   end;
   procedure line1218;
   begin
        l12.b:='    ';
        l12.n:=0;
        l12.o:=':if okay(a) then';
        l13:='        begin';
        l14.a:='              ';
        l14.b:=fname;
        l14.c:=':=true;';
        l15:='               return;';
        l16:='        end;';
        l17.a:='    99:';
        l17.b:=fname;
        l17.c:=':=false';
        l18:='end;';
   end;
```

```
procedure write15;
var i,nc:integer;
begin
    with l1 do
    begin
        writeln(lib4,a:9,n:strlen(str(n)),o:22);
    end;
    with l2 do
    begin
        write(lib4,dec);
        for i:=1 to li do
        begin
            nc:=rl(num(.i.).two);
            write(lib4,num(.i.).two:nc,num(.i.).del:1);
        end;
        writeln(lib4);
    end;
    writeln(lib4,l4);
    with l3 do
    begin
        b:=l2.num(.li-2.).two;
        d:=b;
        nc:=nl(b);
        writeln(lib4,a:34,b:nc,c:8,d:nc,e:6);
    end;
end;
procedure write67;
var nc:integer;
begin
    with l6 do
    begin
    nc:=nl(num1);
    write(lib4,no:4,num1:nc,cond:12);
    nc:=nl(num2);
    write(lib4,name:strlen(str(name)),comma1:1,num2:nc);
    nc:=nl(num3);
    writeln(lib4,comma2:1,a:1,other:13,num3:nc,del:1);
    end;
    writeln(lib4,l7);
end;
procedure write811;
var nc:integer;
begin
    with l8 do
    begin
    nc:=nl(num1);
    write(lib4,no:4,num1:nc,cond:12);
    nc:=nl(num2);
    write(lib4,name:strlen(str(name)),comma1:1,num2:nc);
    nc:=nl(num3);
    writeln(lib4,comma2:1,a:1,other:13,num3:nc,del:1);
    end;
```

```
     with l9 do
     begin
         nc:=nl(num);
         writeln(lib4,f:23,num:nc,other:16);
     end;
     with l10 do
     begin
         nc:=nl(num);
         writeln(lib4,other:13,num:nc,f:2);
     end;
     with l11 do
     begin
         nc:=nl(num);
         writeln(lib4,other:13,num:nc,f:1);
     end;
end;
```

```
procedure write1218;
var nc:integer;
begin
    with l12 do
    begin
        nc:=nl(n);
        writeln(lib4,b:4,n:nc,o:16);
    end;
    writeln(lib4,l13);
    with l14 do
    begin
        writeln(lib4,a:11,b:strlen(str(b)),c:7);
    end;
    writeln(lib4,l15);
    writeln(lib4,l16);
    with l17 do
    begin
        writeln(lib4,a:7,b:strlen(str(b)),c:7);
    end;
    writeln(lib4,l18);
end;
begin
    if(exist(fname)) then return;
    t:=succ(t); ext(.t.):=fname;
    line15;
    aa:=pred(a);
    li:=b-a+2;
    for i:=1 to li do
    begin
        aa:=succ(aa);
        l2.num(.i.).two:=aa;
    end;
    li:=succ(li);
    l2.num(.li.).two:=99;
    l2.num(.li.).del:=';';
    write15;
    line67;
    l6.num1:=a;
    l6.name:=change(proc(.a.).name);
    l6.num2:=a;
    l6.num3:=succ(a);
    write67;
    for i:=succ(a) to b do
    begin
        line811;
        l8.num1:=i;
        l8.name:=change(proc(.i.).name);
        l8.num2:=i;
        l8.num3:=succ(i);
        l9.num:=pred(i);
        l10.num:=pred(i);
        l11.num:=pred(i);
```

```
          write811;
      end;
      line1218;
      l12.n:=succ(b);
      write1218;
  end; (* createrule *)
  begin
      rewrite(lib4,'name=lib4.pascal.a1');
      rewrite(user,'name=user.pascal.a1');
      t:=1; ext(.t.):='  ';
      for i:=1 to ax do
      begin
          pan:=alt(.i.).name;
          p1:=alt(.i.).bbegin;
          p2:=alt(.i.).bend;
          p3:=alt(.i.).yesno;
          if(p1=0) then  createfun(pan);
          if(p1>0) then  createrule(pan,p1,p2);
      end;
      for i:=1 to px do
      begin
          if(proc(.i.).yesno=1) then continue;
          if(proc(.i.).ptype=6) then continue;
          if(proc(.i.).name='query') then qq:=i;
          pan:=proc(.i.).name;
          p1:=proc(.i.).abegin;
          p2:=proc(.i.).aend;
          if(p1>0) then altbody(pan,p1,p2);
      end;
      for i:=1 to px do
      begin
          if(proc(.i.).yesno=1) then continue;
          if(proc(.i.).ptype=6) then continue;
          pan:=proc(.i.).name;
          p1:=proc(.i.).bbegin;
          p2:=proc(.i.).bend;
          if((p1=0) and (proc(.i.).relativity=0))
            then  createfun(pan);
      end;
      for i:=1 to px do
      begin
          if(proc(.i.).yesno=1) then continue;
          if(proc(.i.).ptype=6) then continue;
          pan:=proc(.i.).name;
          p1:=proc(.i.).bbegin;
          p2:=proc(.i.).bend;
          if(p1>0) then  createrule(pan,p1,p2);
      end;
      takelib0;
      takelib2;
      takelib1;
      takelib4;
      takelib3;
  end;  (*create pascal*)
```

```
begin
   cms('exec e',ret);
   datetime(date,time);
   message(' miniprolog  npgs              '||str(date)
   ||str('                       ')
   ||str(time),0);
   message(' ',0);
   checktokens;
   message(' ',0);
   if not(tokenerror) then
      lexicalanalyzer
   else
   begin
       message
('compilation terminated due to user errors.',0);
       message
('virtual compilation time;in microseconds',-clock);
       retcode(-1);
       halt;
   end;
   if not(lexerror) then
   begin
       createarrays;
       if (defined and not(recursive) and query) then
       begin
           message
('no compiler detected errors..source lines',-line);
           createpascal;
       end
       else
       begin
           message
('compilation terminated due to user errors.',0);
           retcode(-1);
           halt;
       end
   end
   else
   begin
       message
('compilation terminated due to user errors.',0);
       retcode(-1);
       halt;
   end;
       message
('virtual compilation time;in  microseconds',-clock);
       retcode(0);
end.
```

APPENDIX C

OBJECT PROGRAM

```
program user(input,output);
const
    pempty=false;
    aempty=false;
    tempty=false;
    px=48;
    ax=44;
    tx=235;
    qq=45;
type
    procrec=record
        rulenum:integer;
        relnum:integer;
        name:alpha;
        ptype:integer;  (* 4..6 *)
        relativity:integer;
        pointer1:integer;
        pointer2:integer;
        bbegin:integer;
        bend:integer;
        abegin:integer;
        aend:integer;
        yesno:integer;
        callee:integer;
        as:integer;
        ae:integer;
        now:integer;
    end;
    parrec=record
        rulenum:integer;
        relnum:integer;
        name:alpha;
        ptype:integer;        (* 7..10. *)
        locality:integer;
        pointer:integer;
        ntype:integer;
        nbind:integer;
        nmatch:boolean;
        who:integer;
    end;
var
    tracefile,term,procfile,paramfile,
    listing,altfile,user,parfile:text;
    trace2:text;
    proc:array(.1..px.) of procrec;
    par:array(.1..tx.) of parrec;
    alt:array(.1..ax.) of procrec;
    cpt:array(.1..tx.) of alpha;
    sign:char;cx,ret,lnum:integer;resatisfy:boolean;
```

```
procedure cms(const parmstr:string; var rc:integer);
    external;
procedure writeit;
var i:integer;
begin
    for i:=1 to tx do
    begin
        write(tracefile,i:2,'.');
        write(tracefile,par(.i.).rulenum:4);
        write(tracefile,par(.i.).relnum:4);
        write(tracefile,' ':4);
        write(tracefile,par(.i.).name:12);
        write(tracefile,par(.i.).ptype:4);
        write(tracefile,par(.i.).locality:4);
        write(tracefile,par(.i.).pointer:4);
        write(tracefile,par(.i.).ntype:4);
        write(tracefile,par(.i.).nbind:4);
        write(tracefile,par(.i.).nmatch:5);
        write(tracefile,par(.i.).who:4);
        writeln(tracefile);
    end;
        writeln(tracefile);
    for i:=1 to px do
    begin
        write(trace2,i:2,'.');
        write(trace2,proc(.i.).rulenum:3);
        write(trace2,proc(.i.).relnum:3);
        write(trace2,' ':4);
        write(trace2,proc(.i.).name:12);
        write(trace2,proc(.i.).ptype:3);
        write(trace2,proc(.i.).relativity:3);
        write(trace2,proc(.i.).pointer1:3);
        write(trace2,proc(.i.).pointer2:3);
        write(trace2,proc(.i.).bbegin:3);
        write(trace2,proc(.i.).bend:3);
        write(trace2,proc(.i.).abegin:3);
        write(trace2,proc(.i.).aend:3);
        write(trace2,proc(.i.).yesno:2);
        write(trace2,proc(.i.).callee:3);
        write(trace2,proc(.i.).as:3);
        write(trace2,proc(.i.).ae:3);
        write(trace2,proc(.i.).now:3);
        writeln(trace2);
     end;
     writeln(trace2);
end;
```

```
procedure message(const msg:string;valint:integer);
var term:text;
begin
    termout(term);
    if (valint>0) then
    begin
       writeln(term,valint:3,str('.    ')||msg);
    end
    else if (valint=0) then
    begin
       writeln(term,msg);
       lnum:=succ(lnum);
       writeln(listing,msg);
    end
    else
    begin
       writeln(term,msg,(-valint));
    end;
    close(term);
end;
```

```
procedure putfiles;
label a1,a2;
var i,f:integer;
    pp:char;
begin
    if tempty then goto a1;
    reset(paramfile);
    for i:=1 to tx do
    begin
        read(paramfile,f);
        read(paramfile,pp);
        read(paramfile,par(.i.).rulenum);
        read(paramfile,par(.i.).relnum);
        read(paramfile,par(.i.).name);
        read(paramfile,par(.i.).ptype);
        read(paramfile,par(.i.).locality);
        read(paramfile,par(.i.).pointer);
        read(paramfile,par(.i.).ntype);
        readln(paramfile,par(.i.).nbind);
        par(.i.).rmatch:=false;
    end;
a1: if pempty then goto a2;
    reset(procfile);
    for i:=1 to px do
    begin
        read(procfile,f);
        read(procfile,pp);
        read(procfile,proc(.i.).rulenum);
        read(procfile,proc(.i.).relnum);
        read(procfile,proc(.i.).name);
        read(procfile,proc(.i.).ptype);
        read(procfile,proc(.i.).relativity);
        read(procfile,proc(.i.).pointer1);
        read(procfile,proc(.i.).pointer2);
        read(procfile,proc(.i.).bbegin);
        read(procfile,proc(.i.).bend);
        read(procfile,proc(.i.).abegin);
        read(procfile,proc(.i.).aend);
        read(procfile,proc(.i.).yesno);
        read(procfile,proc(.i.).callee);
        read(procfile,proc(.i.).as);
        read(procfile,proc(.i.).ae);
        readln(procfile,proc(.i.).now);
    end;
a2: if aempty then return;
    reset(altfile);
    for i:=1 to ax do
    begin
        read(altfile,f);
        read(altfile,pp);
        read(altfile,alt(.i.).rulenum);
        read(altfile,alt(.i.).relnum);
        read(altfile,alt(.i.).name);
        read(altfile,alt(.i.).ptype);
        read(altfile,alt(.i.).relativity);
        read(altfile,alt(.i.).pointer1);
        read(altfile,alt(.i.).pointer2);
        read(altfile,alt(.i.).bbegin);
        read(altfile,alt(.i.).bend);
        read(altfile,alt(.i.).abegin);
        readln(altfile,alt(.i.).aend);
    end;
end; (*putfiles *)
```

```
procedure resetit(a:integer);
begin
    if((a-1)=qq) then return;
    proc(.a.).as:=proc(.a.).abegin;
    proc(.a.).ae:=proc(.a.).aend;
    proc(.a.).now:=proc(.a.).abegin;
    if(proc(.a.).as<>0) then return;
    proc(.a.).as:=1;
    proc(.a.).ae:=1;
    proc(.a.).now:=1;
end; (*resetit *)
function possible(a:integer):boolean;
var a1,a2,a3:boolean;
begin
    a1:=false;
    a2:=false;
    a3:=false;
    a1:=(proc(.a.).now<=proc(.a.).ae);
    if(proc(.a.).ptype=6) then a2:=true;
    if(resatisfy) then a3:=true;
    possible:=a1 or a2 or a3;
end; (*possible*)
```

```
procedure break(a:integer);
label a1;
var i,j,k,call,asta,afin,pbeg,pend:integer;
begin
    if tempty then return;
    if(proc(.a.).ptype=4) then return;
    if(proc(.a.).ptype=6) then
    begin
        pbeg:=proc(.a.).pointer1;
        pend:=proc(.a.).pointer2;
        goto a1;
    end;
    if(proc(.a.).callee>0) then
    begin
        call:=proc(.a.).callee;
        asta:=proc(.call.).pointer1;
        afin:=proc(.call.).pointer2;
    end
    else
    begin
        call:=proc(.a.).now;
        call:=pred(call);
        asta:=proc(.call.).pointer1;
        afin:=proc(.call.).pointer2;
    end;
    pbeg:=proc(.a.).pointer1;
    pend:=proc(.a.).pointer2;
    for i:=asta to afin do
    begin
        if(i=0) then continue;
        if(par(.i.).who<>i) then continue;
        par(.i.).rmatch:=false;
        if(par(.i.).ptype=1) then
        begin
            par(.i.).ntype:=0;
            par(.i.).nbind:=0;
        end;
    end;
a1:
    j:=qq;
    j:=proc(.j.).bend;
    j:=proc(.j.).pointer2;
    for i:=pbeg to pend do
    begin
        if(i=0) then continue;
        if(par(.i.).who<>i) then continue;
        par(.i.).nmatch:=false;
        if(par(.i.).ptype=1) then
        begin
            par(.i.).ntype:=0;
            par(.i.).nbind:=0;
            for k:=pbeg  to j do
            begin
                if(k=0) then continue;
                if(par(.k.).who<>k) then continue;
                if(par(.k.).ptype<>1) then continue;
                if(par(.k.).name<>par(.i.).name)
                   then continue;
                par(.k.).nmatch:=false;
                par(.k.).ntype:=0;
                par(.k.).nbind:=0;
            end;
        end;
    end;
end; (*break*)
```

```
function analyze
        (a:integer; var re,le:integer):boolean;
var f,l,j,m:integer;
    tf:boolean;
    nset:set of 1..4;
begin
    f:=proc(.a.).pointer1;
    l:=proc(.a.).pointer2;
    nset:=(. .);
    for j:=f to l do
    begin
        if(j=0) then continue;
        nset:=nset+(.par(.j.).locality.);
    end;
    for j:=1 to (l-f+1) do
    begin
        if(j in nset) then continue;
        m:=j;
        leave;
    end;
    le:=0;
    analyze:=true;
    for j:=f to l do
    begin
        if(j=0) then continue;
        if(par(.j.).locality<m) then le:=succ(le);
        if(par(.j.).locality>m) then leave;
    end;
    re:=0;
    for j:=f to l do
        if(par(.j.).locality>m) then re:=succ(re);
    assert ((le=1) or (le=3));
    assert ((re=1) or (re=3));
    if((le=1) and (re=1)) then
    begin
        tf:=((par(.f.).ntype<>par(.l.).ntype) or
             (par(.f.).ntype=0) or
             (par(.l.).ntype=0));
    end;
    if((le=1) and (re=3)) then
    begin
        tf:=((par(.f.).ntype<>2) or
             (par(.l.).ntype<>2) or
             (par(.l-2.).ntype<>2));
    end;
    if((le=3) and (re=1)) then
    begin
        tf:=((par(.l.).ntype<>2) or
             (par(.f.).ntype<>2) or
             (par(.f+2.).ntype<>2));
    end;
    if((le=3) and (re=3)) then
    begin
        tf:=((par(.f.).ntype<>2) or
             (par(.f+2.).ntype<>2) or
             (par(.l-2.).ntype<>2) or
             (par(.l.).ntype<>2));
    end;
    if tf then
        analyze:=false;
end; (* analyze *)
```

```
function is(a,k:integer):boolean;
var l,f,i,j,pb,pe:integer;
begin
    f:=proc(.a.).pointer1;
    l:=succ(f);
    par(.f.).ntype:=par(.l.).ntype;
    par(.f.).nbind:=par(.l.).nbind;
    par(.f.).nmatch:=true;
    is:=true;
    if(succ(l)>tx) then return;
    if(succ(a)>px) then return;
    j:=qq;
    pb:=succ(a);
    pb:=proc(.pb.).pointer1;
    pe:=proc(.j.).bend;
    pe:=proc(.pe.).pointer2;
    for i:=pb to pe do
    begin
        if(i=0) then continue;
        if(par(.i.).name<>par(.f.).name)
          then continue;
        if(par(.i.).ptype<>1) then continue;
        par(.i.).ntype:=par(.f.).ntype;
        par(.i.).nbind:=par(.f.).nbind;
        par(.i.).who:=par(.f.).who;
    end;
end; (*is*)
function eval(op1:real;op:integer;op2:real):real;
begin
    case op of
     7:eval:=op1+op2;
     8:eval:=op1-op2;
     9:eval:=op1*op2;
    10:eval:=op1/op2;
    end;
end;
function doreal(p:alpha):real;
var
    num:real;
begin
   readstr(str(p),num);
   doreal:=num;
end;
```

```
function lessthan(a,k:integer):boolean;
var re,le,f,l,p1,p2,p3,p4:integer;
    pa,pb,pc,pd:alpha;op1,op2:integer;
begin
    if not(analyze(a,re,le)) then
    begin
        lesstham:=false; return;
    end;
    f:=proc(.a.).pointer1;
    l:=proc(.a.).pointer2;
    if((re=1) and (le=1)) then
    begin
       if((par(.f.).ntype<>2) or (par(.l.).ntype<>2))
       then begin
           lessthan:=false;return;
       end;
       p1:=par(.f.).nbind;
       pa:=par(.p1.).name;
       p2:=par(.l.).nbind;
       pb:=par(.p2.).name;
       lessthan:=(doreal(pa)<doreal(pb));
       return;end;
    if((re=1) and (le=3)) then
    begin
        p1:=par(.f.).nbind;
        pa:=par(.p1.).name;
        p2:=par(.l-2.).nbind;
        pb:=par(.p2.).name;
        p3:=par(.l.).nbind;
        pc:=par(.p3.).name;
        op1:=par(.l-1.).ptype;
        lessthan:=
        doreal(pa)<eval(doreal(pb),op1,doreal(pc));
        return;end;
    if((re=3) and (le=1)) then
    begin
        p1:=par(.f.).nbind;
        pa:=par(.p1.).name;
        p2:=par(.f+2.).nbind;
        pb:=par(.p2.).name;
        op1:=par(.f+1.).ptype;
        p3:=par(.l.).nbind;
        pc:=par(.p3.).name;
        lessthan:=
        eval(doreal(pa),op1,doreal(pb))<doreal(pc);
        return;end;
    if((re=3) and (le=3)) then
    begin
        p1:=par(.f.).nbind;
        pa:=par(.p1.).name;
        p2:=par(.f+2.).nbind;
        pb:=par(.p2.).name;
        op1:=par(.f+1.).ptype;
        p3:=par(.l-2.).nbind;
        pc:=par(.p3.).name;
        p4:=par(.l.).nbind;
        pd:=par(.p4.).name;
        op2:=par(.l-1.).ptype;
        lessthan:=eval(doreal(pa),op1,doreal(pb))<
                  eval(doreal(pc),op2,doreal(pd));
        return;end;
end; (* lessthan*)
```

```
function notequal(a,k:integer):boolean;
var re,le,f,l,p1,p2,p3,p4:integer;
    pa,pb,pc,pd:alpha;op1,op2:integer;
begin
    if not(analyze(a,re,le)) then
    begin
        notequal:=false;return;
    end;
    f:=proc(.a.).pointer1;
    l:=proc(.a.).pointer2;
    if((re=1) and (le=1)) then
    begin
       if((par(.f.).ntype<>2) or
         (par(.l.).ntype<>2)) then
       begin
            notequal:=false;return;
       end;
       p1:=par(.f.).nbind;
       pa:=par(.p1.).name;
       p2:=par(.l.).nbind;
       pb:=par(.p2.).name;
       notequal:=(doreal(pa)<>doreal(pb));
       return;end;
    if((re=1) and (le=3)) then
    begin
        p1:=par(.f.).nbind;
        pa:=par(.p1.).name;
        p2:=par(.l-2.).nbind;
        pb:=par(.p2.).name;
        p3:=par(.l.).nbind;
        pc:=par(.p3.).name;
        op1:=par(.l-1.).ptype;
        notequal:=
        doreal(pa)<>eval(doreal(pb),op1,doreal(pc));
        return;end;
    if((re=3) and (le=1)) then
    begin
        p1:=par(.f.).nbind;
        pa:=par(.p1.).name;
        p2:=par(.f+2.).nbind;
        pb:=par(.p2.).name;
        op1:=par(.f+1.).ptype;
        p3:=par(.l.).nbind;
        pc:=par(.p3.).name;
        notequal:=
        eval(doreal(pa),op1,doreal(pb))<>doreal(pc);
        return;
    end;
    if((re=3) and (le=3)) then
    begin
        p1:=par(.f.).nbind;
        pa:=par(.p1.).name;
        p2:=par(.f+2.).nbind;
        pb:=par(.p2.).name;
        op1:=par(.f+1.).ptype;
        p3:=par(.l-2.).nbind;
        pc:=par(.p3.).name;
        p4:=par(.l.).nbind;
        pd:=par(.p4.).name;
        op2:=par(.l-1.).ptype;
        notequal:=eval(doreal(pa),op1,doreal(pb))<>
                  eval(doreal(pc),op2,doreal(pd));
        return;
    end;
end; (* notequal *)
```

```
function greatequal(a,k:integer):boolean;
var re,le,f,l,p1,p2,p3,p4:integer;
    pa,pb,pc,pd:alpha;op1,op2:integer;
begin
    if not(analyze(a,re,le)) then
    begin
        greatequal:=false;return;
    end;
    f:=proc(.a.).pointer1;
    l:=proc(.a.).pointer2;
    if((re=1) and (le=1)) then
    begin
        if((par(.f.).ntype<>2) or
            (par(.l.).ntype<>2)) then
        begin
            greatequal:=false;return;
       end;
       p1:=par(.f.).nbind;
       pa:=par(.p1.).name;
       p2:=par(.l.).nbind;
       pb:=par(.p2.).name;
       greatequal:=(doreal(pa)>=doreal(pb));
       return;end;
    if((re=1) and (le=3)) then
    begin
        p1:=par(.f.).nbind;
        pa:=par(.p1.).name;
        p2:=par(.l-2.).nbind;
        pb:=par(.p2.).name;
        p3:=par(.l.).nbind;
        pc:=par(.p3.).name;
        op1:=par(.l-1.).ptype;
        greatequal:=
        doreal(pa)>=eval(doreal(pb),op1,doreal(pc));
        return;end;
    if((re=3) and (le=1)) then
    begin
        p1:=par(.f.).nbind;
        pa:=par(.p1.).name;
        p2:=par(.f+2.).nbind;
        pb:=par(.p2.).name;
        op1:=par(.f+1.).ptype;
        p3:=par(.l.).nbind;
        pc:=par(.p3.).name;
        greatequal:=
        eval(doreal(pa),op1,doreal(pb))>=doreal(pc);
        return;
    end;
    if((re=3) and (le=3)) then
    begin
        p1:=par(.f.).nbind;
        pa:=par(.p1.).name;
        p2:=par(.f+2.).nbind;
        pb:=par(.p2.).name;
        op1:=par(.f+1.).ptype;
        p3:=par(.l-2.).nbind;
        pc:=par(.p3.).name;
        p4:=par(.l.).nbind;
        pd:=par(.p4.).name;
        op2:=par(.l-1.).ptype;
        greatequal:=eval(doreal(pa),op1,doreal(pb))>=
                    eval(doreal(pc),op2,doreal(pd));
        return;end;
end; (* greatequal *)
```

```
function lessequal(a,k:integer):boolean;
var re,le,f,l,p1,p2,p3,p4:integer;
    pa,pb,pc,pd:alpha;op1,op2:integer;
begin
    if not(analyze(a,re,le)) then
    begin
        lessequal:=false;return;
    end;
    f:=proc(.a.).pointer1;
    l:=proc(.a.).pointer2;
    if((re=1) and (le=1)) then
    begin
        if((par(.f.).ntype<>2) or
           (par(.l.).ntype<>2)) then
        begin
            lessequal:=false;
            return;
        end;
       p1:=par(.f.).nbind;
       pa:=par(.p1.).name;
       p2:=par(.l.).nbind;
       pb:=par(.p2.).name;
       lessequal:=(doreal(pa)<=doreal(pb));
       return;end;
    if((re=1) and (le=3)) then
    begin
        p1:=par(.f.).nbind;
        pa:=par(.p1.).name;
        p2:=par(.l-2.).nbind;
        pb:=par(.p2.).name;
        p3:=par(.l.).nbind;
        pc:=par(.p3.).name;
        op1:=par(.l-1.).ptype;
        lessequal:=
        doreal(pa)<=eval(doreal(pb),op1,doreal(pc));
        return;end;
    if((re=3) and (le=1)) then
    begin
        p1:=par(.f.).nbind;
        pa:=par(.p1.).name;
        p2:=par(.f+2.).nbind;
        pb:=par(.p2.).name;
        op1:=par(.f+1.).ptype;
        p3:=par(.l.).nbind;
        pc:=par(.p3.).name;
        lessequal:=
        eval(doreal(pa),op1,doreal(pb))<=doreal(pc);
        return;end;
    if((re=3) and (le=3)) then
    begin
        p1:=par(.f.).nbind;
        pa:=par(.p1.).name;
        p2:=par(.f+2.).nbind;
        pb:=par(.p2.).name;
        op1:=par(.f+1.).ptype;
        p3:=par(.l-2.).nbind;
        pc:=par(.p3.).name;
        p4:=par(.l.).nbind;
        pd:=par(.p4.).name;
        op2:=par(.l-1.).ptype;
        lessequal:=eval(doreal(pa),op1,doreal(pb))<=
                   eval(doreal(pc),op2,doreal(pd));
        return;end;
end; (* lessequal *)
```

```
function greater(a,k:integer):boolean;
var re,le,f,l,p1,p2,p3,p4:integer;
    pa,pb,pc,pd:alpha;op1,op2:integer;
begin
    if not(analyze(a,re,le)) then
    begin
        greater:=false;return;
    end;
    f:=proc(.a.).pointer1;
    l:=proc(.a.).pointer2;
    if((re=1) and (le=1)) then
    begin
        if((par(.f.).ntype<>2)                              or
(par(.l.).ntype<>2))
        then
        begin
            greater:=false;
            return;
       end;
       p1:=par(.f.).nbind;
       pa:=par(.p1.).name;
       p2:=par(.l.).nbind;
       pb:=par(.p2.).name;
       greater:=(doreal(pa)>doreal(pb));
       return;end;
    if((re=1) and (le=3)) then
    begin
        p1:=par(.f.).nbind;
        pa:=par(.p1.).name;
        p2:=par(.l-2.).nbind;
        pb:=par(.p2.).name;
        p3:=par(.l.).nbind;
        pc:=par(.p3.).name;
        op1:=par(.l-1.).ptype;
        greater:=
        doreal(pa)>eval(doreal(pb),op1,doreal(pc));
        return;end;
    if((re=3) and (le=1)) then
    begin
        p1:=par(.f.).nbind;
        pa:=par(.p1.).name;
        p2:=par(.f+2.).nbind;
        pb:=par(.p2.).name;
        op1:=par(.f+1.).ptype;
        p3:=par(.l.).nbind;
        pc:=par(.p3.).name;
        greater:=
        eval(doreal(pa),op1,doreal(pb))>doreal(pc);
        return;end;
    if((re=3) and (le=3)) then
    begin
        p1:=par(.f.).nbind;
        pa:=par(.p1.).name;
        p2:=par(.f+2.).nbind;
        pb:=par(.p2.).name;
        op1:=par(.f+1.).ptype;
        p3:=par(.l-2.).nbind;
        pc:=par(.p3.).name;
        p4:=par(.l.).nbind;
        pd:=par(.p4.).name;
        op2:=par(.l-1.).ptype;
        greater:=eval(doreal(pa),op1,doreal(pb))>
                 eval(doreal(pc),op2,doreal(pd));
        return;end;
end; (* greater *)
```

```
function equal(a,k:integer):boolean;
var re,le,f,l,p1,p2,p3,p4:integer;
    pa,pb,pc,pd:alpha;op1,op2:integer;
begin
    if not(analyze(a,re,le)) then
    begin
       equal:=false;return;
    end;
    f:=proc(.a.).pointer1;
    l:=proc(.a.).pointer2;
    if((re=1) and (le=1)) then
    begin
       p1:=par(.f.).nbind;
       pa:=par(.p1.).name;
       p2:=par(.l.).nbind;
       pb:=par(.p2.).name;
       if((par(.f.).ntype=2) and (par(.l.).ntype=2))
       then equal:=(doreal(pa)=doreal(pb))
       else
       if((par(.f.).ntype=3) and (par(.l.).ntype=3))
       then equal:=(pa=pb)
       else equal:=false;
       return;end;
    if((re=1) and (le=3)) then
    begin
        p1:=par(.f.).nbind;
        pa:=par(.p1.).name;
        p2:=par(.l-2.).nbind;
        pb:=par(.p2.).name;
        p3:=par(.l.).nbind;
        pc:=par(.p3.).name;
        op1:=par(.l-1.).ptype;
        equal:=doreal(pa)
              =eval(doreal(pb),op1,doreal(pc));
        return;end;
    if((re=3) and (le=1)) then
    begin
        p1:=par(.f.).nbind;
        pa:=par(.p1.).name;
        p2:=par(.f+2.).nbind;
        pb:=par(.p2.).name;
        op1:=par(.f+1.).ptype;
        p3:=par(.l.).nbind;
        pc:=par(.p3.).name;
        equal:=eval(doreal(pa),op1,doreal(pb))
              =doreal(pc);
        return;end;
    if((re=3) and (le=3)) then
    begin
        p1:=par(.f.).nbind;
        pa:=par(.p1.).name;
        p2:=par(.f+2.).nbind;
        pb:=par(.p2.).name;
        op1:=par(.f+1.).ptype;
        p3:=par(.l-2.).nbind;
        pc:=par(.p3.).name;
        p4:=par(.l.).nbind;
        pd:=par(.p4.).name;
        op2:=par(.l-1.).ptype;
        equal:=eval(doreal(pa),op1,doreal(pb))=
                  eval(doreal(pc),op2,doreal(pd));
        return;end;
end; (* equal *)
```

```
function okay(a:integer):boolean;
var pb,pe,i:integer;
begin
    if tempty then okay:=true;
    if tempty then return;
    pb:=proc(.qq.).bbegin;
    pe:=proc(.qq.).bend;
    pb:=proc(.pb.).pointer1;
    pe:=proc(.pe.).pointer2;
    for i:=pb to pe do
    begin
        if (i=0) then continue;
        if (par(.i.).ntype=0) then
        begin
            okay:=false;
            return;
        end;
    end;
    okay:=true;
end; (*okay*)
function first(a:integer):integer;
begin
    first:=proc(.a.).now;
end;
function last(a:integer):integer;
begin
    last:=proc(.a.).ae;
end; (*last*)
```

```
function match(a,i:integer):boolean;
var pbeg,pend,lim1,lim2,asta,afin,call:integer;
procedure matchit(asta,pbeg,pend:integer);
var i,j:integer;
begin
    j:=pred(asta);
    for i:=pbeg to pend do
    begin
        if(i=0) then continue;
        j:=succ(j);
        if(j=0) then continue;
        par(.i.).rmatch
        :=((par(.i.).ntype=par(.j.).ntype) and
        (par(.i.).nbind=par(.j.).nbind));
    end;
end; (*matchit*)
procedure bindproc(asta,pbeg,pend:integer);
var i,j,k:integer;
begin
    j:=pred(asta);
    for i:=pbeg to pend do
    begin
        if(i=0) then continue;
        j:=succ(j);
        if(j=0) then continue;
        if(par(.i.).ntype=0) then
        begin
            par(.i.).ntype:=par(.j.).ntype;
            par(.i.).nbind:=par(.j.).nbind;
            par(.i.).who:=i;
            for k:=pbeg to pend do
            begin
                if(k=0) then continue;
                if(par(.k.).ntype>0) then continue;
                if(par(.k.).name<>par(.i.).name)
                then continue;
                par(.k.).ntype:=par(.i.).ntype;
                par(.k.).nbind:=par(.i.).nbind;
                par(.k.).who:=par(.i.).who;
            end;
        end;
    end;
end; (*bindproc*)
```

```
function checkit:boolean;
var i,j:integer; bool:boolean;
begin
    bool:=true;
    for i:=pbeg to pend do
    begin
        if (i=0) then continue;
        bool:=(bool and par(.i.).nmatch);
    end;
    if bool then
    begin
        checkit:=true;
        return;
    end
    else checkit:=false;
    for i:=pbeg to pend do
    begin
        if (i=0) then continue;
        if (par(.i.).ptype<>1) then continue;
        if (par(.i.).who<>i) then continue;
        par(.i.).ptype:=0;
        par(.i.).nbind:=0;
        par(.i.).nmatch:=false;
    end;
    for i:=asta to afin do
    begin
        if (i=0) then continue;
        if (par(.i.).ptype<>1) then continue;
        if (par(.i.).who<>i) then continue;
        par(.i.).ptype:=0;
        par(.i.).nbind:=0;
        par(.i.).nmatch:=false;
    end;
end; (* checkit *)
```

```
procedure rulebind;
var i,j,rbegin,rend:integer;
begin
    j:=succ(a);
    if(j>px) then return;
    rbegin:=proc(.j.).pointer1;
    j:=qq;
    j:=proc(.j.).tend;
    rend:=proc(.j.).pointer2;
    for i:=pbeg tc pend do
    begin
        if(i=0) then continue;
        if(par(.i.).ntype=0) then continue;
        if(par(.i.).ptype<>1) then continue;
        for j:=rbegin to rend do
        begin
            if(j=0) then continue;
            if(par(.j.).ptype<>1) then continue;
            if(par(.j.).name<>par(.i.).name)
              then continue;
            par(.j.).ntype:=par(.i.).ntype;
            par(.j.).nbind:=par(.i.).nbind;
            par(.j.).who:=par(.i.).who;
        end;
    end;
end; (*rulebind*)
```

```
begin
    afin:=0;
    asta:=0;
    pbeg:=proc(.a.).pointer1;
    pend:=proc(.a.).pointer2;
    call:=proc(.a.).callee;
    if (call=0) then
    begin
        if not(aempty) then
        begin
            asta:=alt(.i.).pointer1;
            afin:=alt(.i.).pointer2;
        end
    end
    else
    begin
        asta:=proc(.call.).pointer1;
        afin:=proc(.call.).pointer2;
    end;
    if(pend=pbeg) then
       lim1:=1
    else
       lim1:=pend-pbeg+1;
    if(afin=asta) then
       lim2:=1
    else
       lim2:=afin-asta+1;
    if((pbeg=0) and (pend=0))then
       lim1:=0;
    if((asta=0) and (afin=0))then
       lim2:=0;
    if(lim1<>lim2) then
    begin
        match:=false;
        return;
   end;
   if((lim1=0) and (lim2=0)) then
   begin
       match:=true;
       return;
   end;
   bindproc(asta,pbeg,pend);
   bindproc(pbeg,asta,afin);
   matchit(asta,pbeg,pend);
   matchit(pbeg,asta,afin);
   if checkit then
   begin
       match:=true;
       rulebind;
       return;
   end;
   match:=false;
end; (* match *)
```

```
function findit(p:alpha):bcolean;
var i:integer;
begin
    findit:=false;
    for i:=1 to cx do
    begin
        if(p=cpt(.i.)) then
        begin
            findit:=true;
            return;
        end;
    end;
end;
procedure print;
var i,k,kj,pbeg,pend,pb,pe:integer;
begin
    if tempty then return;
    cx:=1; cpt(.cx.):=' ';
    pb:=proc(.qq.).bbegin;
    pe:=proc(.qq.).bend;
    pbeg:=proc(.pb.).pointer1;
    pend:=proc(.pe.).pointer2;
    if(pend=0) then
    begin
        kj:=0;
        repeat
             kj:=succ(kj);
             pend:=proc(.pe-kj.).pointer2;
        until (pend>0);
    end;
    for i:=pbeg to pend do
    begin
        if (i=0) then continue;
        if (par(.i.).ptype<>1) then continue;
        k:=par(.i.).nbind;
        if(not(findit(par(.i.).name))) then
        begin
            message(str(par(.i.).name)||
            str(' = ')||str(par(.k.).name),0);
            cx:=succ(cx);
            cpt(.cx.):=par(.i.).name;
        end;
    end;
end;
```

```
function accept(function name(a,j:integer)
          :boolean;a,j:integer):boolean;
var i:integer;
begin
    for i:=first(a) to last(a) do
    begin
        if(first(a)>last(a)) then leave;
        if(name(a,i)) then
        begin
            proc(.a.).now:=succ(i);
            accept:=true;
            return;
        end;
    end;
    accept:=false;
    resetit(a);
end; (* accept *)
```

```
function wp_fuel1(a,i:integer):boolean;
begin
    wp_fuel1:=match(a,i);
end;
function wp_fuel2(a,i:integer):boolean;
begin
    wp_fuel2:=match(a,i);
end;
function wp_fuel3(a,i:integer):boolean;
begin
    wp_fuel3:=match(a,i);
end;
function wp_fuel4(a,i:integer):boolean;
begin
    wp_fuel4:=match(a,i);
end;
function wp_fuel5(a,i:integer):boolean;
begin
    wp_fuel5:=match(a,i);
end;
function wp_fuel6(a,i:integer):boolean;
begin
    wp_fuel6:=match(a,i);
end;
function wp_fuel7(a,i:integer):boolean;
begin
    wp_fuel7:=match(a,i);
end;
function wp_fuel8(a,i:integer):boolean;
begin
    wp_fuel8:=match(a,i);
end;
function wp_fuel9(a,i:integer):boolean;
begin
    wp_fuel9:=match(a,i);
end;
```

```
function wp_ammo10(a,i:integer):boolean;
begin
    wp_ammo10:=match(a,i);
end;
function wp_ammo11(a,i:integer):boolean;
begin
    wp_ammo11:=match(a,i);
end;
function wp_ammo12(a,i:integer):boolean;
begin
    wp_ammo12:=match(a,i);
end;
function wp_ammo13(a,i:integer):boolean;
begin
    wp_ammo13:=match(a,i);
end;
function wp_ammo14(a,i:integer):boolean;
begin
    wp_ammo14:=match(a,i);
end;
function wp_ammo15(a,i:integer):boolean;
begin
    wp_ammo15:=match(a,i);
end;
function wp_ammo16(a,i:integer):boolean;
begin
    wp_ammo16:=match(a,i);
end;
function wp_ammo17(a,i:integer):boolean;
begin
    wp_ammo17:=match(a,i);
end;
function wp_ammo18(a,i:integer):boolean;
begin
    wp_ammo18:=match(a,i);
end;
function wp_ammo19(a,i:integer):boolean;
begin
    wp_ammo19:=match(a,i);
end;
function wp_ammo20(a,i:integer):boolean;
begin
    wp_ammo20:=match(a,i);
end;
```

```
function wp_ammo21(a,i:integer):boolean;
begin
    wp_ammo21:=match(a,i);
end;
function wp_ammo22(a,i:integer):boolean;
begin
    wp_ammo22:=match(a,i);
end;
function wp_ammo23(a,i:integer):boolean;
begin
    wp_ammo23:=match(a,i);
end;
function wp_ammo24(a,i:integer):boolean;
begin
    wp_ammo24:=match(a,i);
end;
function wp_ammo25(a,i:integer):boolean;
begin
    wp_ammo25:=match(a,i);
end;
function wp_ammo26(a,i:integer):boolean;
begin
    wp_ammo26:=match(a,i);
end;
function wp_ammo27(a,i:integer):boolean;
begin
    wp_ammo27:=match(a,i);
end;
function wp_ammo28(a,i:integer):boolean;
begin
    wp_ammo28:=match(a,i);
end;
function wp_ammo29(a,i:integer):boolean;
begin
    wp_ammo29:=match(a,i);
end;
function wp_ammo30(a,i:integer):boolean;
begin
    wp_ammo30:=match(a,i);
end;
function wp_ammo31(a,i:integer):boolean;
begin
    wp_ammo31:=match(a,i);
end;
```

```
function wp_num32(a,i:integer):boolean;
begin
    wp_num32:=match(a,i);
end;
function wp_num33(a,i:integer):boolean;
begin
    wp_num33:=match(a,i);
end;
function wp_num34(a,i:integer):boolean;
begin
    wp_num34:=match(a,i);
end;
function wp_num35(a,i:integer):boolean;
begin
    wp_num35:=match(a,i);
end;
function wp_num36(a,i:integer):boolean;
begin
    wp_num36:=match(a,i);
end;
function wp_num37(a,i:integer):boolean;
begin
    wp_num37:=match(a,i);
end;
function wp_num38(a,i:integer):boolean;
begin
    wp_num38:=match(a,i);
end;
function wp_num39(a,i:integer):boolean;
begin
    wp_num39:=match(a,i);
end;
function wp_num40(a,i:integer):boolean;
begin
    wp_num40:=match(a,i);
end;
function wp_num41(a,i:integer):boolean;
begin
    wp_num41:=match(a,i);
end;
function wp_num42(a,i:integer):boolean;
begin
    wp_num42:=match(a,i);
end;
function wp_num43(a,i:integer):boolean;
begin
    wp_num43:=match(a,i);
end;
function wp_num44(a,i:integer):boolean;
begin
    wp_num44:=match(a,i);
end;
```

```
function wp_fuel(a,i:integer):boolean;
begin
    case i of
    1:wp_fuel:=wp_fuel1(a,i);
    2:wp_fuel:=wp_fuel2(a,i);
    3:wp_fuel:=wp_fuel3(a,i);
    4:wp_fuel:=wp_fuel4(a,i);
    5:wp_fuel:=wp_fuel5(a,i);
    6:wp_fuel:=wp_fuel6(a,i);
    7:wp_fuel:=wp_fuel7(a,i);
    8:wp_fuel:=wp_fuel8(a,i);
    9:wp_fuel:=wp_fuel9(a,i);
    end;
end;
```

```
functicn wp_ammo(a,i:integer):boolean;
begin
    case i of
    10:wp_ammo:=wp_ammo10(a,i);
    11:wp_ammo:=wp_ammo11(a,i);
    12:wp_ammo:=wp_ammo12(a,i);
    13:wp_ammo:=wp_ammo13(a,i);
    14:wp_ammo:=wp_ammo14(a,i);
    15:wp_ammo:=wp_ammo15(a,i);
    16:wp_ammo:=wp_ammo16(a,i);
    17:wp_ammo:=wp_ammo17(a,i);
    18:wp_ammo:=wp_ammo18(a,i);
    19:wp_ammo:=wp_ammo19(a,i);
    20:wp_ammo:=wp_ammo20(a,i);
    21:wp_ammo:=wp_ammo21(a,i);
    22:wp_ammo:=wp_ammo22(a,i);
    23:wp_ammo:=wp_ammo23(a,i);
    24:wp_ammo:=wp_ammo24(a,i);
    25:wp_ammo:=wp_ammo25(a,i);
    26:wp_ammo:=wp_ammo26(a,i);
    27:wp_ammo:=wp_ammo27(a,i);
    28:wp_ammo:=wp_ammo28(a,i);
    29:wp_ammo:=wp_ammo29(a,i);
    30:wp_ammo:=wp_ammo30(a,i);
    31:wp_ammo:=wp_ammo31(a,i);
    end;
end;
```

```
function wp_num(a,i:integer):boolean;
begin
    case i of
    32:wp_num:=wp_num32(a,i);
    33:wp_num:=wp_num33(a,i);
    34:wp_num:=wp_num34(a,i);
    35:wp_num:=wp_num35(a,i);
    36:wp_num:=wp_num36(a,i);
    37:wp_num:=wp_num37(a,i);
    38:wp_num:=wp_num38(a,i);
    39:wp_num:=wp_num39(a,i);
    40:wp_num:=wp_num40(a,i);
    41:wp_num:=wp_num41(a,i);
    42:wp_num:=wp_num42(a,i);
    43:wp_num:=wp_num43(a,i);
    44:wp_num:=wp_num44(a,i);
    end;
end;
```

```
function query(a,i:integer):boolean;
label 46,47,48,49,99;
begin
    if resatisfy then begin break(48); goto 48; end;
    46:if (accept(wp_fuel,46,a)) then goto 47;
       goto 99;
    47:if (accept(wp_ammo,47,a)) then goto 48;
       if not(possible(46)) then goto 99;
       break(46);
       goto  46;
    48:if (accept(wp_num,48,a)) then goto 49;
       if not(possible(47)) then goto 99;
       break(47);
       goto  47;
    49:if okay(a) then
       begin
           query:=true;
           return;
       end;
    99:query:=false
end;
begin
    lnum:=0;
    putfiles;
    message(' execution begins....',0);
    resatisfy:=false;
    sign:=';';
    while (sign=';') do
    begin
        if query(gg,1) then
        begin
            message(' yes',0);
            print; termin(term);
            readln(term,sign);  close(term);
            if(sign=';') then
               resatisfy:=true
            else resatisfy:=false;
            if(lnum>50) then
            begin
                lnum:=0;
                writeln(listing,'1':1);
            end;
            if resatisfy then
               cms('clrscrn',ret);
            if resatisfy then
               message(' resatisfying goal....',0)
            else message(' execution ends....',0);
            continue;
        end;
        message(str(' no'),0);
        message(' execution ends....',0);
        halt;
    end;
end. (* main *)
```

APPENDIX D

PROCEDURE TABLE

1.	1	1	wp_fuel	5	0	1	5	0	0	0	0	1	1	1	1	1
2.	2	1	wp_fuel	5	0	6	10	0	0	0	0	1	1	1	1	1
3.	3	1	wp_fuel	5	0	11	15	0	0	0	0	1	1	1	1	1
4.	4	1	wp_fuel	5	0	16	20	0	0	0	0	1	1	1	1	1
5.	5	1	wp_fuel	5	0	21	25	0	0	0	0	1	1	1	1	1
6.	6	1	wp_fuel	5	0	26	30	0	0	0	0	1	1	1	1	1
7.	7	1	wp_fuel	5	0	31	35	0	0	0	0	1	1	1	1	1
8.	8	1	wp_fuel	5	0	36	40	0	0	0	0	1	1	1	1	1
9.	9	1	wp_fuel	5	0	41	45	0	0	0	0	1	1	1	1	1
10.	10	1	wp_ammo	5	0	46	50	0	0	0	0	1	1	1	1	2
11.	11	1	wp_ammo	5	0	51	55	0	0	0	0	1	1	1	1	2
12.	12	1	wp_ammo	5	0	56	60	0	0	0	0	1	1	1	1	2
13.	13	1	wp_ammo	5	0	61	65	0	0	0	0	1	1	1	1	2
14.	14	1	wp_ammo	5	0	66	70	0	0	0	0	1	1	1	1	2
15.	15	1	wp_ammo	5	0	71	75	0	0	0	0	1	1	1	1	2
16.	16	1	wp_ammo	5	0	76	80	0	0	0	0	1	1	1	1	2
17.	17	1	wp_ammo	5	0	81	85	0	0	0	0	1	1	1	1	2
18.	18	1	wp_ammo	5	0	86	90	0	0	0	0	1	1	1	1	2
19.	19	1	wp_ammo	5	0	91	95	0	0	0	0	1	1	1	1	2
20.	20	1	wp_ammo	5	0	96	100	0	0	0	0	1	1	1	1	2
21.	21	1	wp_ammo	5	0	101	105	0	0	0	0	1	1	1	1	2
22.	22	1	wp_ammo	5	0	106	110	0	0	0	0	1	1	1	1	2
23.	23	1	wp_ammo	5	0	111	115	0	0	0	0	1	1	1	1	2
24.	24	1	wp_ammo	5	0	116	120	0	0	0	0	1	1	1	1	2
25.	25	1	wp_ammo	5	0	121	125	0	0	0	0	1	1	1	1	2
26.	26	1	wp_ammo	5	0	126	130	0	0	0	0	1	1	1	1	2
27.	27	1	wp_ammo	5	0	131	135	0	0	0	0	1	1	1	1	2
28.	28	1	wp_ammo	5	0	136	140	0	0	0	0	1	1	1	1	2
29.	29	1	wp_ammo	5	0	141	145	0	0	0	0	1	1	1	1	2
30.	30	1	wp_ammo	5	0	146	150	0	0	0	0	1	1	1	1	2
31.	31	1	wp_ammo	5	0	151	155	0	0	0	0	1	1	1	1	2
32.	32	1	wp_num	5	0	156	160	0	0	0	0	1	1	1	1	3
33.	33	1	wp_num	5	0	161	165	0	0	0	0	1	1	1	1	3
34.	34	1	wp_num	5	0	166	170	0	0	0	0	1	1	1	1	3
35.	35	1	wp_num	5	0	171	175	0	0	0	0	1	1	1	1	3
36.	36	1	wp_num	5	0	176	180	0	0	0	0	1	1	1	1	3
37.	37	1	wp_num	5	0	181	185	0	0	0	0	1	1	1	1	3
38.	38	1	wp_num	5	0	186	190	0	0	0	0	1	1	1	1	3
39.	39	1	wp_num	5	0	191	195	0	0	0	0	1	1	1	1	3
40.	40	1	wp_num	5	0	196	200	0	0	0	0	1	1	1	1	3
41.	41	1	wp_num	5	0	201	205	0	0	0	0	1	1	1	1	3
42.	42	1	wp_num	5	0	206	210	0	0	0	0	1	1	1	1	3
43.	43	1	wp_num	5	0	211	215	0	0	0	0	1	1	1	1	3
44.	44	1	wp_num	5	0	216	220	0	0	0	0	1	1	1	1	3
45.	45	1	query	4	0	0	0	46	48	0	0	0	1	1	1	4
46.	45	3	wp_fuel	5	1	221	225	0	0	1	9	0	1	9	1	1
47.	45	16	wp_ammo	5	2	226	230	0	0	10	31	0	10	31	10	2
48.	45	29	wp_num	5	3	231	235	0	0	32	44	0	32	44	32	3

APPENDIX E

PARAMETERS TABLE

1.	1	3	tk	3	1	1	3	1
2.	1	5	1	2	2	1	2	2
3.	1	7	f1	3	3	1	3	3
4.	1	9	5CC	2	4	1	2	4
5.	1	11	50C0	2	5	1	2	5
6.	2	3	fac	3	1	2	3	6
7.	2	5	1	2	2	2	2	2
8.	2	7	f2	3	3	2	3	8
9.	2	9	800	2	4	2	2	9
10.	2	11	40C0	2	5	2	2	10
11.	3	3	atgm	3	1	3	3	11
12.	3	5	1	2	2	3	2	2
13.	3	7	f3	3	3	3	3	13
14.	3	9	40C	2	4	3	2	14
15.	3	11	3000	2	5	3	2	15
16.	4	3	helo	3	1	4	3	16
17.	4	5	1	2	2	4	2	2
18.	4	7	f2	3	3	4	3	8
19.	4	9	250	2	4	4	2	19
20.	4	11	10C0	2	5	4	2	20
21.	5	3	tk	3	1	5	3	1
22.	5	5	2	2	2	5	2	22
23.	5	7	f4	3	3	5	3	23
24.	5	9	6CC	2	4	5	2	24
25.	5	11	5500	2	5	5	2	25
26.	6	3	fac	3	1	6	3	6
27.	6	5	2	2	2	6	2	22
28.	6	7	f2	3	3	6	3	8
29.	6	9	900	2	4	6	2	29
30.	6	11	60C0	2	5	6	2	30
31.	7	3	atgm	3	1	7	3	11
32.	7	5	2	2	2	7	2	22
33.	7	7	f5	3	3	7	3	33
34.	7	9	50C	2	4	7	2	4
35.	7	11	3500	2	5	7	2	35
36.	8	3	helo	3	1	8	3	16
37.	8	5	2	2	2	8	2	22
38.	8	7	f3	3	3	8	3	13
39.	8	9	350	2	4	8	2	39
40.	8	11	20C0	2	5	8	2	40
41.	9	3	atgm	3	1	9	3	11
42.	9	5	3	2	2	9	2	42
43.	9	7	f6	3	3	9	3	43
44.	9	9	45C	2	4	9	2	44
45.	9	11	3800	2	5	9	2	45
46.	10	3	tk	3	1	10	3	1
47.	10	5	1	2	2	10	2	2
48.	10	7	1	2	3	10	2	2
49.	10	9	3	2	4	10	2	42
50.	10	11	a1	3	5	10	3	50

51.	11	3	tk	3	1	11	3	1
52.	11	5	1	2	2	11	2	2
53.	11	7	1	2	3	11	2	2
54.	11	9	3	2	4	11	2	42
55.	11	11	a2	3	5	11	3	55
56.	12	3	tk	3	1	12	3	1
57.	12	5	1	2	2	12	2	2
58.	12	7	2	2	3	12	2	22
59.	12	9	3	2	4	12	2	42
60.	12	11	a1	3	5	12	3	50
61.	13	3	tk	3	1	13	3	1
62.	13	5	1	2	2	13	2	2
63.	13	7	2	2	3	13	2	22
64.	13	9	3	2	4	13	2	42
65.	13	11	a2	3	5	13	3	55
66.	14	3	fac	3	1	14	3	6
67.	14	5	1	2	2	14	2	2
68.	14	7	1	2	3	14	2	2
69.	14	9	4	2	4	14	2	69
70.	14	11	a6	3	5	14	3	70
71.	15	3	fac	3	1	15	3	6
72.	15	5	1	2	2	15	2	2
73.	15	7	1	2	3	15	2	2
74.	15	9	4	2	4	15	2	69
75.	15	11	a3	3	5	15	3	75
76.	16	3	atgm	3	1	16	3	11
77.	16	5	1	2	2	16	2	2
78.	16	7	1	2	3	16	2	2
79.	16	9	3	2	4	16	2	42
80.	16	11	a4	3	5	16	3	80
81.	17	3	helo	3	1	17	3	16
82.	17	5	1	2	2	17	2	2
83.	17	7	2	2	3	17	2	22
84.	17	9	2	2	4	17	2	22
85.	17	11	a4	3	5	17	3	80
86.	18	3	helo	3	1	18	3	16
87.	18	5	1	2	2	18	2	2
88.	18	7	2	2	3	18	2	22
89.	18	9	2	2	4	18	2	22
90.	18	11	a3	3	5	18	3	75
91.	19	3	tk	3	1	19	3	1
92.	19	5	2	2	2	19	2	22
93.	19	7	3	2	3	19	2	42
94.	19	9	3	2	4	19	2	42
95.	19	11	a1	3	5	19	3	50
96.	20	3	tk	3	1	20	3	1
97.	20	5	2	2	2	20	2	22
98.	20	7	3	2	3	20	2	42
99.	20	9	3	2	4	20	2	42

100.	20	11	a3	3	5	20	3	75
101.	21	3	belo	3	1	21	3	16
102.	21	5	2	2	2	21	2	22
103.	21	7	3	2	3	21	2	42
104.	21	9	3	2	4	21	2	42
105.	21	11	a6	3	5	21	3	70
106.	22	3	belo	3	1	22	3	16
107.	22	5	2	2	2	22	2	22
108.	22	7	3	2	3	22	2	42
109.	22	9	3	2	4	22	2	42
110.	22	11	a3	3	5	22	3	75
111.	23	3	atgm	3	1	23	3	11
112.	23	5	2	2	2	23	2	22
113.	23	7	3	2	3	23	2	42
114.	23	9	3	2	4	23	2	42
115.	23	11	a5	3	5	23	3	115
116.	24	3	fac	3	1	24	3	6
117.	24	5	2	2	2	24	2	22
118.	24	7	3	2	3	24	2	42
119.	24	9	4	2	4	24	2	69
120.	24	11	a6	3	5	24	3	70
121.	25	3	tk	3	1	25	3	1
122.	25	5	2	2	2	25	2	22
123.	25	7	2	2	3	25	2	22
124.	25	9	3	2	4	25	2	42
125.	25	11	a1	3	5	25	3	50
126.	26	3	tk	3	1	26	3	1
127.	26	5	3	2	2	26	2	42
128.	26	7	5	2	3	26	2	128
129.	26	9	3	2	4	26	2	42
130.	26	11	a3	3	5	26	3	75
131.	27	3	atgm	3	1	27	3	11
132.	27	5	3	2	2	27	2	42
133.	27	7	5	2	3	27	2	128
134.	27	9	3	2	4	27	2	42
135.	27	11	a4	3	5	27	3	80
136.	28	3	fac	3	1	28	3	6
137.	28	5	1	2	2	28	2	2
138.	28	7	5	2	3	28	2	128
139.	28	9	2	2	4	28	2	22
140.	28	11	a6	3	5	28	3	70
141.	29	3	fac	3	1	29	3	6
142.	29	5	1	2	2	29	2	2
143.	29	7	5	2	3	29	2	128
144.	29	9	2	2	4	29	2	22
145.	29	11	a3	3	5	29	3	75
146.	30	3	tk	3	1	30	3	1
147.	30	5	1	2	2	30	2	2
148.	30	7	5	2	3	30	2	128
149.	30	9	3	2	4	30	2	42
150.	30	11	a1	3	5	30	3	50

151.	31	3	tk	3	1	31	3	1
152.	31	5	1	2	2	31	2	2
153.	31	7	5	2	3	31	2	128
154.	31	9	3	2	4	31	2	42
155.	31	11	a3	3	5	31	3	75
156.	32	3	tk	3	1	32	3	1
157.	32	5	1	2	2	32	2	2
158.	32	7	1	2	3	32	2	2
159.	32	9	3	2	4	32	2	42
160.	32	11	40	2	5	32	2	160
161.	33	3	tk	3	1	33	3	1
162.	33	5	1	2	2	33	2	2
163.	33	7	2	2	3	33	2	22
164.	33	9	3	2	4	33	2	42
165.	33	11	50	2	5	33	2	165
166.	34	3	fac	3	1	34	3	6
167.	34	5	1	2	2	34	2	2
168.	34	7	1	2	3	34	2	2
169.	34	9	4	2	4	34	2	69
170.	34	11	5	2	5	34	2	128
171.	35	3	atgm	3	1	35	3	11
172.	35	5	1	2	2	35	2	2
173.	35	7	1	2	3	35	2	2
174.	35	9	3	2	4	35	2	42
175.	35	11	20	2	5	35	2	175
176.	36	3	helo	3	1	36	3	16
177.	36	5	1	2	2	36	2	2
178.	36	7	2	2	3	36	2	22
179.	36	9	2	2	4	36	2	22
180.	36	11	8	2	5	36	2	180
181.	37	3	tk	3	1	37	3	1
182.	37	5	2	2	2	37	2	22
183.	37	7	3	2	3	37	2	42
184.	37	9	3	2	4	37	2	42
185.	37	11	28	2	5	37	2	185
186.	38	3	helo	3	1	38	3	16
187.	38	5	2	2	2	38	2	22
188.	38	7	3	2	3	38	2	42
189.	38	9	3	2	4	38	2	42
190.	38	11	12	2	5	38	2	190
191.	39	3	atgm	3	1	39	3	11
192.	39	5	2	2	2	39	2	22
193.	39	7	3	2	3	39	2	42
194.	39	9	3	2	4	39	2	42
195.	39	11	22	2	5	39	2	195
196.	40	3	fac	3	1	40	3	6
197.	40	5	2	2	2	40	2	22
198.	40	7	3	2	3	40	2	42
199.	40	9	4	2	4	40	2	69
200.	40	11	8	2	5	40	2	180

201.	41	3	tk	3	1	41	3	1
202.	41	5	2	2	2	41	2	22
203.	41	7	2	2	3	41	2	22
204.	41	9	3	2	4	41	2	42
205.	41	11	42	2	5	41	2	205
206.	42	3	atgm	3	1	42	3	11
207.	42	5	3	2	2	42	2	42
208.	42	7	5	2	3	42	2	128
209.	42	9	3	2	4	42	2	42
210.	42	11	18	2	5	42	2	210
211.	43	3	fac	3	1	43	3	6
212.	43	5	1	2	2	43	2	2
213.	43	7	5	2	3	43	2	128
214.	43	9	2	2	4	43	2	22
215.	43	11	8	2	5	43	2	180
216.	44	3	tk	3	1	44	3	1
217.	44	5	1	2	2	44	2	2
218.	44	7	5	2	3	44	2	128
219.	44	9	3	2	4	44	2	42
220.	44	11	48	2	5	44	2	220
221.	45	5	helo	3	1	46	3	16
222.	45	7	WTYPE	1	2	46	0	0
223.	45	9	WFTYPE	1	3	46	0	0
224.	45	11	WFCAP	1	4	46	0	0
225.	45	13	WACAP	1	5	46	0	0
226.	45	18	helo	3	1	47	3	16
227.	45	20	WTYPE	1	2	47	0	0
228.	45	22	IX	1	3	47	0	0
229.	45	24	IY	1	4	47	0	0
230.	45	26	WATYPE	1	5	47	0	0
231.	45	31	helo	3	1	48	3	16
232.	45	33	WTYPE	1	2	48	0	0
233.	45	35	IX	1	3	48	0	0
234.	45	37	IY	1	4	48	0	0
235.	45	39	WNUM	1	5	48	0	0

APPENDIX F

ALTERNATIVE CLAUSES TABLE

1.	1	1	wp_fuel1	5	0	1	5	0	0	1	1
2.	2	1	wp_fuel2	5	0	6	10	0	0	2	2
3.	3	1	wp_fuel3	5	0	11	15	0	0	3	3
4.	4	1	wp_fuel4	5	0	16	20	0	0	4	4
5.	5	1	wp_fuel5	5	0	21	25	0	0	5	5
6.	6	1	wp_fuel6	5	0	26	30	0	0	6	6
7.	7	1	wp_fuel7	5	0	31	35	0	0	7	7
8.	8	1	wp_fuel8	5	0	36	40	0	0	8	8
9.	9	1	wp_fuel9	5	0	41	45	0	0	9	9
10.	10	1	wp_ammo10	5	0	46	50	0	0	10	10
11.	11	1	wp_ammo11	5	0	51	55	0	0	11	11
12.	12	1	wp_ammo12	5	0	56	60	0	0	12	12
13.	13	1	wp_ammo13	5	0	61	65	0	0	13	13
14.	14	1	wp_ammo14	5	0	66	70	0	0	14	14
15.	15	1	wp_ammo15	5	0	71	75	0	0	15	15
16.	16	1	wp_ammo16	5	0	76	80	0	0	16	16
17.	17	1	wp_ammo17	5	0	81	85	0	0	17	17
18.	18	1	wp_ammo18	5	0	86	90	0	0	18	18
19.	19	1	wp_ammo19	5	0	91	95	0	0	19	19
20.	20	1	wp_ammo20	5	0	96	100	0	0	20	20
21.	21	1	wp_ammo21	5	0	101	105	0	0	21	21
22.	22	1	wp_ammo22	5	0	106	110	0	0	22	22
23.	23	1	wp_ammo23	5	0	111	115	0	0	23	23
24.	24	1	wp_ammo24	5	0	116	120	0	0	24	24
25.	25	1	wp_ammo25	5	0	121	125	0	0	25	25
26.	26	1	wp_ammo26	5	0	126	130	0	0	26	26
27.	27	1	wp_ammo27	5	0	131	135	0	0	27	27
28.	28	1	wp_ammo28	5	0	136	140	0	0	28	28
29.	29	1	wp_ammo29	5	0	141	145	0	0	29	29
30.	30	1	wp_ammo30	5	0	146	150	0	0	30	30
31.	31	1	wp_ammo31	5	0	151	155	0	0	31	31
32.	32	1	wp_num32	5	0	156	160	0	0	32	32
33.	33	1	wp_num33	5	0	161	165	0	0	33	33
34.	34	1	wp_num34	5	0	166	170	0	0	34	34
35.	35	1	wp_num35	5	0	171	175	0	0	35	35
36.	36	1	wp_num36	5	0	176	180	0	0	36	36
37.	37	1	wp_num37	5	0	181	185	0	0	37	37
38.	38	1	wp_num38	5	0	186	190	0	0	38	38
39.	39	1	wp_num39	5	0	191	195	0	0	39	39
40.	40	1	wp_num40	5	0	196	200	0	0	40	40
41.	41	1	wp_num41	5	0	201	205	0	0	41	41
42.	42	1	wp_num42	5	0	206	210	0	0	42	42
43.	43	1	wp_num43	5	0	211	215	0	0	43	43
44.	44	1	wp_num44	5	0	216	220	0	0	44	44

APPENDIX G

SAMPLE PROGRAM 1

```
wp_fuel(tk  ,1,f1,500,5000).
wp_fuel(fac ,1,f2,800,4000).
wp_fuel(atgm,1,f3,400,3000).
wp_fuel(helo,1,f2,250,1000).
wp_fuel(tk  ,2,f4,600,5500).
wp_fuel(fac ,2,f2,900,6000).
wp_fuel(atgm,2,f5,500,3500).
wp_fuel(helo,2,f3,350,2000).
wp_fuel(atgm,3,f6,450,3800).
wp_ammo(tk  ,1,1,3,a1).
wp_ammo(tk  ,1,1,3,a2).
wp_ammo(tk  ,1,2,3,a1).
wp_ammo(tk  ,1,2,3,a2).
wp_ammo(fac ,1,1,4,a6).
wp_ammo(fac ,1,1,4,a3).
wp_ammo(atgm,1,1,3,a4).
wp_ammo(helo,1,2,2,a4).
wp_ammo(helo,1,2,2,a3).
wp_ammo(tk  ,2,3,3,a1).
wp_ammo(tk  ,2,3,3,a3).
wp_ammo(helo,2,3,3,a6).
wp_ammo(helo,2,3,3,a3).
wp_ammo(atgm,2,3,3,a5).
wp_ammo(fac ,2,3,4,a6).
wp_ammo(tk  ,2,2,3,a1).
wp_ammo(tk  ,3,5,3,a3).
wp_ammo(atgm,3,5,3,a4).
wp_ammo(fac ,1,5,2,a6).
wp_ammo(fac ,1,5,2,a3).
wp_ammo(tk  ,1,5,3,a1).
wp_ammo(tk  ,1,5,3,a3).
wp_num(tk  ,1,1,3,40).
wp_num(tk  ,1,2,3,50).
wp_num(fac ,1,1,4, 5).
wp_num(atgm,1,1,3,20).
wp_num(helo,1,2,2, 8).
wp_num(tk  ,2,3,3,28).
wp_num(helo,2,3,3,12).
wp_num(atgm,2,3,3,22).
wp_num(fac ,2,3,4, 8).
wp_num(tk  ,2,2,3,42).
wp_num(atgm,3,5,3,18).
wp_num(fac ,1,5,2, 8).
wp_num(tk  ,1,5,3,48).
query:-wp_fuel(helo,WTYPE,WFTYPE,WFCAP,WACAP),
       wp_ammo(helo,WTYPE,LX,LY,WATYPE),
       wp_num(helc,WTYPE,LX,LY,WNUM).
```

APPENDIX H

OUTPUT OF PROGRAM 1

```
EXECUTION BEGINS....
yes
   WTYPE        =     1
   WFTYPE       =     f2
   WFCAP        =     250
   WACAP        =     1000
   LX           =     2
   LY           =     2
   WATYPE       =     a4
   WNUM         =     8
RESATISFYING GOAL....
yes
   WTYPE        =     1
   WFTYPE       =     f2
   WFCAP        =     250
   WACAP        =     1000
   LX           =     2
   LY           =     2
   WATYPE       =     a3
   WNUM         =     8
RESATISFYING GOAL....
yes
   WTYPE        =     2
   WFTYPE       =     f3
   WFCAP        =     350
   WACAP        =     2000
   LX           =     3
   LY           =     3
   WATYPE       =     a6
   WNUM         =     12
RESATISFYING GOAL....
yes
   WTYPE        =     2
   WFTYPE       =     f3
   WFCAP        =     350
   WACAP        =     2000
   LX           =     3
   LY           =     3
   WATYPE       =     a3
   WNUM         =     12
RESATISFYING GOAL....
no
EXECUTION ENDS....
```

APPENDIX I

SAMPLE PROGRAM 2

```
wp_fuel(tk  ,1,f1,500,5000).
wp_fuel(fac ,1,f2,800,4000).
wp_fuel(atgm,1,f3,400,3000).
wp_fuel(helo,1,f2,250,1000).
wp_fuel(tk  ,2,f4,600,5500).
wp_fuel(fac ,2,f2,900,6000).
wp_fuel(atgm,2,f5,500,3500).
wp_fuel(helo,2,f3,350,2000).
wp_fuel(atgm,3,f6,450,3800).
wp_ammo(tk  ,1,1,3,a1).
wp_ammo(tk  ,1,1,3,a2).
wp_ammo(tk  ,1,2,3,a1).
wp_ammo(tk  ,1,2,3,a2).
wp_ammo(fac ,1,1,4,a6).
wp_ammo(fac ,1,1,4,a3).
wp_ammo(atgm,1,1,3,a4).
wp_ammo(helo,1,2,2,a4).
wp_ammo(helo,1,2,2,a3).
wp_ammo(tk  ,2,3,3,a1).
wp_ammo(tk  ,2,3,3,a3).
wp_ammo(helo,2,3,3,a6).
wp_ammo(helo,2,3,3,a3).
wp_ammo(atgm,2,3,3,a5).
wp_ammo(fac ,2,3,4,a6).
wp_ammo(tk  ,2,2,3,a1).
wp_ammo(tk  ,3,5,3,a3).
wp_ammo(atgm,3,5,3,a4).
wp_ammo(fac ,1,5,2,a6).
wp_ammo(fac ,1,5,2,a3).
wp_ammo(tk  ,1,5,3,a1).
wp_ammo(tk  ,1,5,3,a3).
wp_num(tk  ,1,1,3,40).
wp_num(tk  ,1,2,3,50).
wp_num(fac ,1,1,4, 5).
wp_num(atgm,1,1,3,20).
wp_num(helo,1,2,2, 8).
wp_num(tk  ,2,3,3,28).
wp_num(helo,2,3,3,12).
wp_num(atgm,2,3,3,22).
wp_num(fac ,2,3,4, 8).
wp_num(tk  ,2,2,3,42).
wp_num(atgm,3,5,3,18).
wp_num(fac ,1,5,2, 8).
wp_num(tk  ,1,5,3,48).
query:-wp_fuel(WCLASS,WTYPE,WFTYPE,WFCAP,WACAP),
       WFCAP<=800,WFCAP>=450,
       wp_ammo(WCLASS,WTYPE,LX,LY,WATYPE).
```

APPENDIX J

OUTPUT OF PROGRAM 2

```
EXECUTION BEGINS....
yes
   WCLASS        =     tk
   WTYPE         =     1
   WFTYPE        =     f1
   WFCAP         =     500
   WACAP         =     5000
   LX            =     1
   LY            =     3
   WATYPE        =     a1
RESATISFYING GOAL....
yes
   WCLASS        =     tk
   WTYPE         =     1
   WFTYPE        =     f1
   WFCAP         =     500
   WACAP         =     5000
   LX            =     1
   LY            =     3
   WATYPE        =     a2
RESATISFYING GOAL....
yes
   WCLASS        =     tk
   WTYPE         =     1
   WFTYPE        =     f1
   WFCAP         =     500
   WACAP         =     5000
   LX            =     2
   LY            =     3
   WATYPE        =     a1
RESATISFYING GOAL....
yes
   WCLASS        =     tk
   WTYPE         =     1
   WFTYPE        =     f1
   WFCAP         =     500
   WACAP         =     5000
   LX            =     2
   LY            =     3
   WATYPE        =     a2
RESATISFYING GOAL....
yes
   WCLASS        =     tk
   WTYPE         =     1
   WFTYPE        =     f1
   WFCAP         =     500
   WACAP         =     5000
   LX            =     5
   LY            =     3
   WATYPE        =     a1
```

```
RESATISFYING GOAL....
yes
    WCLASS         =     tk
    WTYPE          =     1
    WFTYPE         =     f1
    WFCAP          =     500
    WACAP          =     5000
    LX             =     5
    LY             =     3
    WATYPE         =     a3
RESATISFYING GOAL....
yes
    WCLASS         =     fac
    WTYPE          =     1
    WFTYPE         =     f2
    WFCAP          =     800
    WACAP          =     4000
    LX             =     1
    LY             =     4
    WATYPE         =     a6
RESATISFYING GOAL....
yes
    WCLASS         =     fac
    WTYPE          =     1
    WFTYPE         =     f2
    WFCAP          =     800
    WACAP          =     4000
    LX             =     1
    LY             =     4
    WATYPE         =     a3
RESATISFYING GOAL....
yes
    WCLASS         =     fac
    WTYPE          =     1
    WFTYPE         =     f2
    WFCAP          =     800
    WACAP          =     4000
    LX             =     5
    LY             =     2
    WATYPE         =     a6
RESATISFYING GOAL....
yes
    WCLASS         =     fac
    WTYPE          =     1
    WFTYPE         =     f2
    WFCAP          =     800
    WACAP          =     4000
    LX             =     5
    LY             =     2
    WATYPE         =     a3
```

```
RESATISFYING GOAL....
yes
   WCLASS         =     tk
   WTYPE          =     2
   WFTYPE         =     f4
   WFCAP          =     600
   WACAP          =     5500
   LX             =     3
   LY             =     3
   WATYPE         =     a1
RESATISFYING GOAL....
yes
   WCLASS         =     tk
   WTYPE          =     2
   WFTYPE         =     f4
   WFCAP          =     600
   WACAP          =     5500
   LX             =     3
   LY             =     3
   WATYPE         =     a3
RESATISFYING GOAL....
yes
   WCLASS         =     tk
   WTYPE          =     2
   WFTYPE         =     f4
   WFCAP          =     600
   WACAP          =     5500
   LX             =     2
   LY             =     3
   WATYPE         =     a1
RESATISFYING GOAL....
yes
   WCLASS         =     atgm
   WTYPE          =     2
   WFTYPE         =     f5
   WFCAP          =     500
   WACAP          =     3500
   LX             =     3
   LY             =     3
   WATYPE         =     a5
RESATISFYING GOAL....
yes
   WCLASS         =     atgm
   WTYPE          =     3
   WFTYPE         =     f6
   WFCAP          =     450
   WACAP          =     3800
   LX             =     5
   LY             =     3
   WATYPE         =     a4
RESATISFYING GOAL....
no
EXECUTION ENDS....
```

APPENDIX K

SAMPLE PROGRAM 3

```
wp_fuel(tk  ,1,f1,500,5000).
wp_fuel(fac ,1,f2,800,4000).
wp_fuel(atgm,1,f3,400,3000).
wp_fuel(helo,1,f2,250,1000).
wp_fuel(tk  ,2,f4,600,5500).
wp_fuel(fac ,2,f2,900,6000).
wp_fuel(atgm,2,f5,500,3500).
wp_fuel(helo,2,f3,350,2000).
wp_fuel(atgm,3,f6,450,3800).
wp_ammo(tk  ,1,1,3,a1).
wp_ammo(tk  ,1,1,3,a2).
wp_ammo(tk  ,1,2,3,a1).
wp_ammo(tk  ,1,2,3,a2).
wp_ammo(fac ,1,1,4,a6).
wp_ammo(fac ,1,1,4,a3).
wp_ammo(atgm,1,1,3,a4).
wp_ammo(helo,1,2,2,a4).
wp_ammo(helo,1,2,2,a3).
wp_ammo(tk  ,2,3,3,a1).
wp_ammo(tk  ,2,3,3,a3).
wp_ammo(helo,2,3,3,a6).
wp_ammo(helo,2,3,3,a3).
wp_ammo(atgm,2,3,3,a5).
wp_ammo(fac ,2,3,4,a6).
wp_ammo(tk  ,2,2,3,a1).
wp_ammo(tk  ,3,5,3,a3).
wp_ammo(atgm,3,5,3,a4).
wp_ammo(fac ,1,5,2,a6).
wp_ammo(fac ,1,5,2,a3).
wp_ammo(tk  ,1,5,3,a1).
wp_ammo(tk  ,1,5,3,a3).
wp_num(tk  ,1,1,3,40).
wp_num(tk  ,1,2,3,50).
wp_num(fac ,1,1,4, 5).
wp_num(atgm,1,1,3,20).
wp_num(helo,1,2,2, 8).
wp_num(tk  ,2,3,3,28).
wp_num(helo,2,3,3,12).
wp_num(atgm,2,3,3,22).
wp_num(fac ,2,3,4, 8).
wp_num(tk  ,2,2,3,42).
wp_num(atgm,3,5,3,18).
wp_num(fac ,1,5,2, 8).
wp_num(tk  ,1,5,3,48).
query:-wp_fuel(fac,WTYPE,WFTYPE,WFCAP,WACAP),
       WFCAP<=900,WFCAP>=800,
       WACAP<=5500,WACAP>=4000,
       wp_ammo(fac,WTYPE,LX,LY,WATYPE).
```

APPENDIX L

OUTPUT OF PROGRAM 3

```
EXECUTION BEGINS....
yes
   WTYPE        =    1
   WFTYPE       =    f2
   WFCAP        =    800
   WACAP        =    4000
   LX           =    1
   LY           =    4
   WATYPE       =    a6
RESATISFYING GOAL....
yes
   WTYPE        =    1
   WFTYPE       =    f2
   WFCAP        =    800
   WACAP        =    4000
   LX           =    1
   LY           =    4
   WATYPE       =    a3
RESATISFYING GOAL....
yes
   WTYPE        =    1
   WFTYPE       =    f2
   WFCAP        =    800
   WACAP        =    4000
   LX           =    5
   LY           =    2
   WATYPE       =    a6
RESATISFYING GOAL....
yes
   WTYPE        =    1
   WFTYPE       =    f2
   WFCAP        =    800
   WACAP        =    4000
   LX           =    5
   LY           =    2
   WATYPE       =    a3
RESATISFYING GOAL....
no
EXECUTION ENDS....
```

APPENDIX M

SAMPLE PROGRAM 4

```
wp_fuel(tk  ,1,f1,500,5000).
wp_fuel(fac ,1,f2,800,4000).
wp_fuel(atgm,1,f3,400,3000).
wp_fuel(helo,1,f2,250,1000).
wp_fuel(tk  ,2,f4,600,5500).
wp_fuel(fac ,2,f2,900,6000).
wp_fuel(atgm,2,f5,500,3500).
wp_fuel(helo,2,f3,350,2000).
wp_fuel(atgm,3,f6,450,3800).
wp_ammo(tk  ,1,1,3,a1).
wp_ammo(tk  ,1,1,3,a2).
wp_ammo(tk  ,1,2,3,a1).
wp_ammo(tk  ,1,2,3,a2).
wp_ammo(fac ,1,1,4,a6).
wp_ammo(fac ,1,1,4,a3).
wp_ammo(atgm,1,1,3,a4).
wp_ammo(helo,1,2,2,a4).
wp_ammo(helo,1,2,2,a3).
wp_ammo(tk  ,2,3,3,a1).
wp_ammo(tk  ,2,3,3,a3).
wp_ammo(helo,2,3,3,a6).
wp_ammo(helo,2,3,3,a3).
wp_ammo(atgm,2,3,3,a5).
wp_ammo(fac ,2,3,4,a6).
wp_ammo(tk  ,2,2,3,a1).
wp_ammo(tk  ,3,5,3,a3).
wp_ammo(atgm,3,5,3,a4).
wp_ammo(fac ,1,5,2,a6).
wp_ammo(fac ,1,5,2,a3).
wp_ammo(tk  ,1,5,3,a1).
wp_ammo(tk  ,1,5,3,a3).
wp_num(tk  ,1,1,3,40).
wp_num(tk  ,1,2,3,50).
wp_num(fac ,1,1,4, 5).
wp_num(atgm,1,1,3,20).
wp_num(helo,1,2,2, 8).
wp_num(tk  ,2,3,3,28).
wp_num(helo,2,3,3,12).
wp_num(atgm,2,3,3,22).
wp_num(fac ,2,3,4, 8).
wp_num(tk  ,2,2,3,42).
wp_num(atgm,3,5,3,18).
wp_num(fac ,1,5,2, 8).
wp_num(tk  ,1,5,3,48).
query:-wp_ammo(WCLASS,WTYPE,LX,LY,WATYPE),
       wp_num(WCLASS,WTYPE,LX,LY,WNUM).
```

APPENDIX N

OUTPUT OF PROGRAM 4

```
EXECUTION BEGINS....
yes
    WCLASS          =     tk
    WTYPE           =     1
    LX              =     1
    LY              =     3
    WATYPE          =     a1
    WNUM            =     40
RESATISFYING GOAL....
yes
    WCLASS          =     tk
    WTYPE           =     1
    LX              =     1
    LY              =     3
    WATYPE          =     a2
    WNUM            =     40
RESATISFYING GOAL....
yes
    WCLASS          =     tk
    WTYPE           =     1
    LX              =     2
    LY              =     3
    WATYPE          =     a1
    WNUM            =     50
RESATISFYING GOAL....
yes
    WCLASS          =     tk
    WTYPE           =     1
    LX              =     2
    LY              =     3
    WATYPE          =     a2
    WNUM            =     50
RESATISFYING GOAL....
yes
    WCLASS          =     fac
    WTYPE           =     1
    LX              =     1
    LY              =     4
    WATYPE          =     a6
    WNUM            =     5
RESATISFYING GOAL....
yes
    WCLASS          =     fac
    WTYPE           =     1
    LX              =     1
    LY              =     4
    WATYPE          =     a3
    WNUM            =     5
```

```
RESATISFYING GOAL....
yes
    WCLASS         =     atgm
    WTYPE          =     1
    LX             =     1
    LY             =     3
    WATYPE         =     a4
    WNUM           =     20
RESATISFYING GOAL....
yes
    WCLASS         =     helo
    WTYPE          =     1
    LX             =     2
    LY             =     2
    WATYPE         =     a4
    WNUM           =     8
RESATISFYING GOAL....
yes
    WCLASS         =     helo
    WTYPE          =     1
    LX             =     2
    LY             =     2
    WATYPE         =     a3
    WNUM           =     8
RESATISFYING GOAL....
yes
    WCLASS         =     tk
    WTYPE          =     2
    LX             =     3
    LY             =     3
    WATYPE         =     a1
    WNUM           =     28
RESATISFYING GOAL....
yes
    WCLASS         =     tk
    WTYPE          =     2
    LX             =     3
    LY             =     3
    WATYPE         =     a3
    WNUM           =     28
RESATISFYING GOAL....
yes
    WCLASS         =     helo
    WTYPE          =     2
    LX             =     3
    LY             =     3
    WATYPE         =     a6
    WNUM           =     12
```

```
RESATISFYING GOAL....
yes
    WCLASS          =     helo
    WTYPE           =     2
    LX              =     3
    LY              =     3
    WATYPE          =     a3
    WNUM            =     12
RESATISFYING GOAL....
yes
    WCLASS          =     atgm
    WTYPE           =     2
    LX              =     3
    LY              =     3
    WATYPE          =     a5
    WNUM            =     22
RESATISFYING GOAL....
yes
    WCLASS          =     fac
    WTYPE           =     2
    LX              =     3
    LY              =     4
    WATYPE          =     a6
    WNUM            =     8
RESATISFYING GOAL....
yes
    WCLASS          =     tk
    WTYPE           =     2
    LX              =     2
    LY              =     3
    WATYPE          =     a1
    WNUM            =     42
RESATISFYING GOAL....
yes
    WCLASS          =     atgm
    WTYPE           =     3
    LX              =     5
    LY              =     3
    WATYPE          =     a4
    WNUM            =     18
RESATISFYING GOAL....
yes
    WCLASS          =     fac
    WTYPE           =     1
    LX              =     5
    LY              =     2
    WATYPE          =     a6
    WNUM            =     8
```

```
RESATISFYING GOAL....
yes
    WCLASS       =     fac
    WTYPE        =     1
    LX           =     5
    LY           =     2
    WATYPE       =     a3
    WNUM         =     8
RESATISFYING GOAL....
yes
    WCLASS       =     tk
    WTYPE        =     1
    LX           =     5
    LY           =     3
    WATYPE       =     a1
    WNUM         =     48
RESATISFYING GOAL....
yes
    WCLASS       =     tk
    WTYPE        =     1
    LX           =     5
    LY           =     3
    WATYPE       =     a3
    WNUM         =     48
RESATISFYING GOAL....
no
EXECUTION ENDS....
```

APPENDIX O

SAMPLE PROGRAM 5

```
wp_fuel(tk  ,1,f1,500,5000).
wp_fuel(fac ,1,f2,800,4000).
wp_fuel(atgm,1,f3,400,3000).
wp_fuel(helo,1,f2,250,1000).
wp_fuel(tk  ,2,f4,600,5500).
wp_fuel(fac ,2,f2,900,6000).
wp_fuel(atgm,2,f5,500,3500).
wp_fuel(helo,2,f3,350,2000).
wp_fuel(atgm,3,f6,450,3800).
wp_ammo(tk  ,1,1,3,a1).
wp_ammo(tk  ,1,1,3,a2).
wp_ammo(tk  ,1,2,3,a1).
wp_ammo(tk  ,1,2,3,a2).
wp_ammo(fac ,1,1,4,a6).
wp_ammo(fac ,1,1,4,a3).
wp_ammo(atgm,1,1,3,a4).
wp_ammo(helo,1,2,2,a4).
wp_ammo(helo,1,2,2,a3).
wp_ammo(tk  ,2,3,3,a1).
wp_ammo(tk  ,2,3,3,a3).
wp_ammo(helo,2,3,3,a6).
wp_ammo(helo,2,3,3,a3).
wp_ammo(atgm,2,3,3,a5).
wp_ammo(fac ,2,3,4,a6).
wp_ammo(tk  ,2,2,3,a1).
wp_ammo(tk  ,3,5,3,a3).
wp_ammo(atgm,3,5,3,a4).
wp_ammo(fac ,1,5,2,a6).
wp_ammo(fac ,1,5,2,a3).
wp_ammo(tk  ,1,5,3,a1).
wp_ammo(tk  ,1,5,3,a3).
wp_num(tk  ,1,1,3,40).
wp_num(tk  ,1,2,3,50).
wp_num(fac ,1,1,4, 5).
wp_num(atgm,1,1,3,20).
wp_num(helo,1,2,2, 8).
wp_num(tk  ,2,3,3,28).
wp_num(helo,2,3,3,12).
wp_num(atgm,2,3,3,22).
wp_num(fac ,2,3,4, 8).
wp_num(tk  ,2,2,3,42).
wp_num(atgm,3,5,3,18).
wp_num(fac ,1,5,2, 8).
wp_num(tk  ,1,5,3,48).
query:-wp_ammo(WCLASS,WTYPE,LX,LY,WATYPE),
       wp_num(WCLASS,WTYPE,LX,LY,WNUM).
```

APPENDIX P

OUTPUT OF PROGRAM 5

```
EXECUTION BEGINS....
yes
   WCLASS        =     tk
   WTYPE         =     1
   LX            =     1
   LY            =     3
   WATYPE        =     a1
   WNUM          =     40
RESATISFYING GOAL....
yes
   WCLASS        =     tk
   WTYPE         =     1
   LX            =     1
   LY            =     3
   WATYPE        =     a2
   WNUM          =     40
RESATISFYING GOAL....
yes
   WCLASS        =     tk
   WTYPE         =     1
   LX            =     2
   LY            =     3
   WATYPE        =     a1
   WNUM          =     50
RESATISFYING GOAL....
yes
   WCLASS        =     tk
   WTYPE         =     1
   LX            =     2
   LY            =     3
   WATYPE        =     a2
   WNUM          =     50
RESATISFYING GOAL....
yes
   WCLASS        =     fac
   WTYPE         =     1
   LX            =     1
   LY            =     4
   WATYPE        =     a6
   WNUM          =     5
RESATISFYING GOAL....
yes
   WCLASS        =     fac
   WTYPE         =     1
   LX            =     1
   LY            =     4
   WATYPE        =     a3
   WNUM          =     5
```

```
RESATISFYING GOAL....
yes
    WCLASS          =     atgm
    WTYPE           =     1
    LX              =     1
    LY              =     3
    WATYPE          =     a4
    WNUM            =     20
RESATISFYING GOAL....
yes
    WCLASS          =     helo
    WTYPE           =     1
    LX              =     2
    LY              =     2
    WATYPE          =     a4
    WNUM            =     8
RESATISFYING GOAL....
yes
    WCLASS          =     helo
    WTYPE           =     1
    LX              =     2
    LY              =     2
    WATYPE          =     a3
    WNUM            =     8
RESATISFYING GOAL....
yes
    WCLASS          =     tk
    WTYPE           =     2
    LX              =     3
    LY              =     3
    WATYPE          =     a1
    WNUM            =     28
RESATISFYING GOAL....
yes
    WCLASS          =     tk
    WTYPE           =     2
    LX              =     3
    LY              =     3
    WATYPE          =     a3
    WNUM            =     28
RESATISFYING GOAL....
yes
    WCLASS          =     helo
    WTYPE           =     2
    LX              =     3
    LY              =     3
    WATYPE          =     a6
    WNUM            =     12
```

```
RESATISFYING GOAL....
yes
    WCLASS          =      helo
    WTYPE           =      2
    LX              =      3
    LY              =      3
    WATYPE          =      a3
    WNUM            =      12
RESATISFYING GOAL....
yes
    WCLASS          =      atgm
    WTYPE           =      2
    LX              =      3
    LY              =      3
    WATYPE          =      a5
    WNUM            =      22
RESATISFYING GOAL....
yes
    WCLASS          =      fac
    WTYPE           =      2
    LX              =      3
    LY              =      4
    WATYPE          =      a6
    WNUM            =      8
RESATISFYING GOAL....
yes
    WCLASS          =      tk
    WTYPE           =      2
    LX              =      2
    LY              =      3
    WATYPE          =      a1
    WNUM            =      42
RESATISFYING GOAL....
yes
    WCLASS          =      atgm
    WTYPE           =      3
    LX              =      5
    LY              =      3
    WATYPE          =      a4
    WNUM            =      18
RESATISFYING GOAL....
yes
    WCLASS          =      fac
    WTYPE           =      1
    LX              =      5
    LY              =      2
    WATYPE          =      a6
    WNUM            =      8
```

```
RESATISFYING GOAL....
yes
   WCLASS        =     fac
   WTYPE         =     1
   LX            =     5
   LY            =     2
   WATYPE        =     a3
   WNUM          =     8
RESATISFYING GOAL....
yes
   WCLASS        =     tk
   WTYPE         =     1
   LX            =     5
   LY            =     3
   WATYPE        =     a1
   WNUM          =     48
RESATISFYING GOAL....
yes
   WCLASS        =     tk
   WTYPE         =     1
   LX            =     5
   LY            =     3
   WATYPE        =     a3
   WNUM          =     48
RESATISFYING GOAL....
no
EXECUTION ENDS....
```

LIST OF REFERENCES

1. Backus, J., "Can Programming Be Liberated from the von Neuman Style?", Communications of the ACM, Vol. 21, 8, pp. 613-615, August 1978.

2. Hoare, C. A. R., "The Emperor's Old Clothes", Communications of the ACM, Vol. 24, 2, pp. 37-38, February 1981.

3. Maclennan, B. J., Functional Programming Methodology, Theory and Practice, Naval Postgraduate School, Monterey, California, (unpublished lecture notes).

4. Moto-oka, T., etal, Fifth Generation Computer Systems, pp. 519-164, North-Holland Publishing Company, New York, 1982.

5. Shapiro, E. Y., "The Fifth Generation Project - A Trip Report", Communications of the ACM, Vol. 26, 9, pp. 637-641, September 1983.

6. Warren, D. H. D., "Efficient Processing of Interactive Relational Database Queries", pp. 272-279, Department of Artificial Intelligence, University of Edinburgh, England.

7. Maclennan, B. J., Principles of Programming Languages, pp. 499-520, CBS College Publishing, New York, 1983.

8. Warren, D. H. D., "Logic Programming and Compiler Writing", pp. 121-124, Department of Artificial Intelligence, University of Edinburgh, England.

9. Warren, D. H. D., "PROLOG - The Language and Its Implementation Compared with Lisp", pp. 109-115, Department of Artificial Intelligence, University of Edinburgh, England.

10. Bruynooghe, M., "The Memory Management of Prolog Implementations", pp. 83-85, Katholieke Universiteit, Heverlee, Belgium.

BIBLIOGRAPHY

Barrett, W. A. and Couch, J. D., Compiler Construction: Theory and Practice, Science Research Associates, Inc., New York, 1979.

Booch, G., Software Engineering with ADA, Cumming Publishing Company, 1983.

Clocksin, W. F. and Mellish, C. S., Programming in Prolog, Springer-Verlag, Berlin, 1981.

Hansson, A., etal, Properties of a Logic Programming Language, Computing Science Department, Uppsala University, Uppsala, Sweden.

Hoare, C. A. R., Hints on Programming Language Design, SIGACT/SIGPLAN Symposium, Boston, 1973.

Hoffman, M. C. and O'Donnel, M. J.,"Programming with Equations", ACM Transactions on Programming Languages and Systems, Vol. 4, 1, January 1982.

Hogger, C. J., Concurrent Logic Programming, Department of Civil Engineering, Imperial College, London, England.

Ince, D. C., "Module Interconnection Languages and Prolog", ACM SIGPLAN Notices, Vol. 19, 8, August 1984.

Jensen, K. and Wirth, N., Pascal User Manual and Report, Springer-Verlag, New York, 1978.

Kowalski, R., "Algorithm = Logic + Control", Communications of the ACM, Vol. 22, 7, July 1979.

O'Keefe, R. A., "Prolog Compared with Lisp", ACM SIGPLAN Notices, Vol. 18, 5, May 1984.

Pereira, L. and Porto, A., Selective Backtracking, Departamento de Informatica, Universidada Nova de Lisboa, Portugal.

Rowe, L. A., "Programming Languages Issues for the 1980's", ACM SIGPLAN Notices, Vol. 19, 8, August 1984.

Santane-Toth, E., etal, Prolog Applications in Hungary, Institute for Co-ordination of Computer Techniques, Budapest, Hungary.

Takeuchi, I., etal, "TAO, A Harmonic Mean of Lisp, Prolog, and Smalltalk", ACM SIGPLAN Notices, Vol. 18, 7, July 1984.

Winograd, T., "Beyond Programming Languages", Communications of the ACM, Vol. 22, 7, July 1979.

Wise, M. J., "EPILOG = PROLOG + Dataflow", ACM SIGPLAN Notices, Vol. 17, 12, December 1982.

Zemrowski, K. M., "Differences Between ANS and ISO Standards for Pascal", ACM SIGPLAN Notices, Vol. 19, 8, August 1984.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943-5100	2
3.	Ms. Debbie Walker, Code 0143C Naval Postgraduate School Monterey, California 93943-5100	1
4.	Ahmet Saraydin Ziyaurrahman Cad. Kelebek Sok. No: 25/8 Ankara, Turkey	2
5.	Computer Technology Programs Curricular Office, Code 37 Naval Postgraduate School Monterey, California 93943-5100	1

215207
Thesis
S16661 Saraydin
c.1
An implementation in Pascal: translation of Prolog into Pascal.